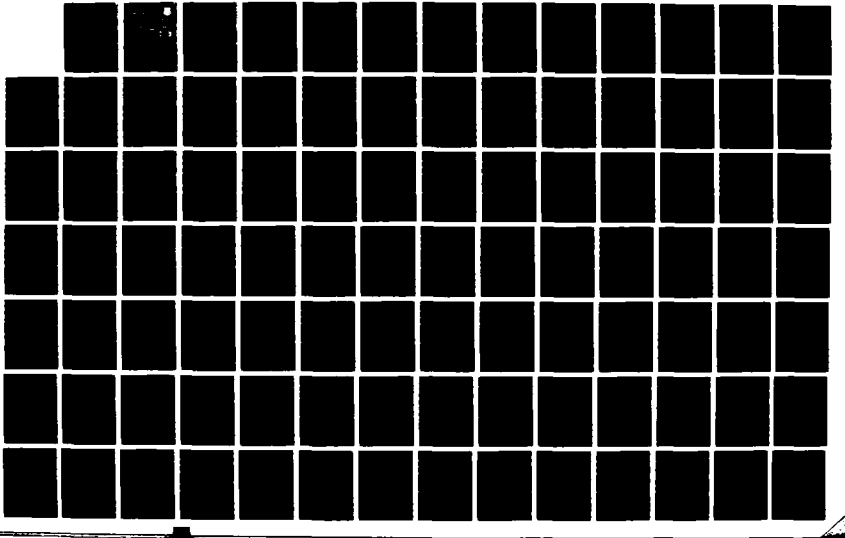
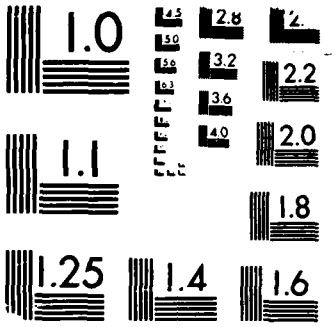
AD-A194 104 ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM 1/3
VOLUME 1(U) SANDERS ASSOCIATES INC NASHUA NH
C BARDAWIL ET AL. DEC 87 RADC-TR-87-249-VOL-1
UNCLASSIFIED F30602-85-C-0254 F/G 12/5 NL





RADC-TR-87-249, Vol I (of two)
Final Technical Report
December 1987

ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM

Sanders Associates, Inc.

Carol Bardawil, Larry Fry, Sandy King, Linda Leszcynski and Graham O'Neil

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-87-249, Vol I (of two) has been reviewed and is approved for publication.

APPROVED: Richard M. Evans

RICHARD M. EVANS
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-87-249, Vol I (of two)
6a. NAME OF PERFORMING ORGANIZATION Sanders Associates, Inc. — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c. ADDRESS (City, State, and ZIP Code) 95 Canal Street, CS 2004 Nashua NH 03061-2004	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center — 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0254
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
63728F	2532	01	16

11. TITLE (Include Security Classification)
ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM

12. PERSONAL AUTHOR(S)
Carol Bardawil, Larry Fry, Sandy King, Linda Leszcynski, Graham O'Neil

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Aug 85 TO Aug 87	December 1987	218

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Software development process, software acquisition model, knowledge-based sytems, documentation standards.
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The goal of this research was to evaluate the software development process for artificial intelligence (AI) systems and postulate a software acquisition model. To accomplish this research, the major elements performed were a literature search, a case study analysis of 26 knowledge based system (KBS) development efforts, and consultation with experienced AI system developers. The results of this study are presented in a two volume report. Volume I presents observations made during the analysis of KBS software developments and provides summaries of the case study data. A comparison of the KBS development process to DOD-STD-2167 is also documented. Volume II discusses a KBS process model and customer/developer interface model. A comparison of the postulated model with DOD-STD-2167 and DOD-STD-2167A (draft) is made in terms of activities, products, reviews and baselines.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL RICHARD M. EVANS	22b. TELEPHONE (Include Area Code) (315) 330-3564 — 22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 — Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Contents

Volume I

List of Tables

Volume I

Contents

Volume II

List of Figures

Volume II

List of Tables

Volume II

List of Contributers

The following people have contributed to the Artificial Intelligence Software Acquisition Program (AISAP) study and final report.

Case Study Participants

Marilyn Aglubat	Northrop Avionics Division
Sam Ashby	Boeing Military Airplane Company
Virginia Barker	Digital Equipment Corporation
Carlos Bhola	Expert Technologies, Inc.
Dr. R. P. Bonasso	The MITRE Corporation (McLean, VA)
Rodney M. Bond	ARINC Research Corporation
Douglas Clafin	Lockheed Aircraft Service Company
Stan Coffman	Lockheed-Georgia Company
P. R. Deweese	Lockheed-Georgia Company
Linda Dudding	Lockheed Aircraft Service Company
Vicki Florian	Software Architecture and Engineering, Inc.
Kimberly Freitas	Inference Corporation
Kermit Gates	PAR Government Systems Corporation
Terry Ginn	Sanders Associates, Inc.
Gary G. Greenfield	Frey Associates, Inc.
Carl Gunther	Inference Corporation
William B. Harrelson	Brattle Research Corporation
David Harris	Sanders Associates, Inc.
Dr. D. F. Hubbard	Carnegie Group, Inc.
Ted Jardine	Boeing Computer Services
Elizabeth Kooker	IBM Federal Systems Group
Robert Lough	Northrop Avionics Division
Jim Montague	Texas Instruments, Inc.
Edward Orciuch	Digital Equipment Corporation
Laurent Piketty	Inference Corporation
M. J. Prelle	The MITRE Corporation (Bedford, MA)
Jack Rahaim	Digital Equipment Corporation
Ethan Scarl	The MITRE Corporation (Bedford, MA)
Anthony D. Vanker	GTE Data Services

Reviewers/Consultants

Dick Cloutier	Sanders Associates, Inc.
Terry Ginn	Sanders Associates, Inc.
David Harris	Sanders Associates, Inc.
Dr. Charles Rich	Massachusetts Institute of Technology
Tom Royer	Sanders Associates, Inc.

Preface

In August 1985, the Rome Air Development Center selected Sanders Associates, Inc. to evaluate the software development process for Artificial Intelligence (AI) systems and postulate a software acquisition model. To accomplish these objectives, Sanders devised a strategy consisting of the following major elements:

- Literature review;
- Case study analysis; and
- Consultation with experienced AI system developers.

The case study analyses represent historical data on 26 knowledge base system (KBS) development efforts. Because the case data focuses on KBS software, the acquisition model developed pertains to KBS efforts.

The results of this study are presented in a two volume report. Volume I presents observations made during the analysis of KBS software developments as well as summaries of the case study data. A comparison of the KBS development process to DOD-STD-2167 is also made.

Volume II presents a KBS process model as well as a postulated customer/developer interface model. A comparison of the postulated model with DOD-STD-2167 and DOD-STD-2167A (draft) is made in terms of activities, products, reviews and baselines.

SECTION 1

Introduction

1.1 Problem Definition

To date, many Artificial Intelligence (AI) systems have been developed in university and other research environments with relatively few oriented towards Department of Defense(DOD) applications. The scale and complexity of these existing systems is substantially less than that anticipated for the Strategic Defense Initiative (SDI) Battle Management/Command Control and Communications (BM/C^3) Technology Program. In addition, the manner in which these AI systems have been developed is largely foreign to the DOD software acquisition process (e.g. DOD-STD-2167) . For instance, conventional software issues such as design reviews, programming languages, documentation, quality assurance procedures, testing and other methods commonly used in military system acquisition are nonexistent, inappropriate or radically different in the AI software development process. Because DOD envisions a need for AI system technology to address many components of the SDI, a clearer understanding of these differences is required. Namely, new policies, procedures, standards and contracting mechanisms must be in place to ensure a high success rate in acquiring AI systems in general. Consequently, specific stated areas of interest to DOD are:

- Analysis of the AI software development process to determine unique characteristics and needs;
- Development of a model(s) for AI software acquisition that satisfies both military needs and unique AI requirements. The model should at least address the following areas:
  - documentation standards;
  - review procedures;
  - rigorous testing methods; and
  - contract mechanisms.
- Analysis and specification for interfacing/integrating AI and conventional software languages and processes.

The Rome Air Development Center (RADC), Air Force Systems Command, at Griffiss Air Force Base, N.Y., was tasked by DOD to respond to SDI needs concerning technology critical to BM/C^3. Having solicited public response in the form of proposals, RADC selected Sanders Associates, Inc to perform and report on a specific study consistent with DOD interests as stated above.

1.2 Solution Strategy

To address the DOD needs as delineated in the previous section, Sanders proposed a two phase approach. Phase I focused on an analysis of the AI software development process The analysis

consisted of collecting case study data for AI systems which had either completed development or were well underway towards completing a prototype. During the Phase I data collection activity, it was learned that case study data was only readily available for Knowledge Based Systems (KBS) and not generally available for other AI application areas (e.g. signal processing, natural language processing, etc.). Because of this, the scope of the study concentrated on KBS. Phase II focused on the development of a KBS software acquisition model. Issues germane to KBS and conventional software interfaces/integration are considered to be a part of both phases.

In Phase I, the analysis effort included an extensive literature search/review, detailed case study evaluations and an assessment of DOD-STD-2167. The basis of the adopted strategy was:

- Extract from the published literature as much information as possible concerning KBS software development techniques and characteristics;
- Design a comprehensive questionnaire to extract data from experienced KBS software builders on specific systems;
- Consult in-house expert system builders to enhance the understanding of the material reviewed;
- Study DOD-STD-2167 methodology for use as the basis for defining deviations appropriate for KBS software development.

This approach enabled the study team to meet the requirements of Phase I. Specifically, KBS software development characteristics were defined and deviations from the conventional software development process were identified.

The Phase I approach also resulted in the collection of data pertinent to the Phase II effort. Namely, case study analyses, particularly in the area of expert systems, led to the identification of KBS areas critical to the SDI effort. Case study participants also provided substantial information concerning "lessons learned" which proved to be invaluable in terms of defining a standard approach that avoids the pitfalls already experienced by others.

Specific goals for Phase II, the final phase of the software project, were to define a model that specified acquisition and development approaches for Knowledge Based System applications. The models developed define the activities, products, reviews and baselines for the development of KBS software. They also identify management needs for visibility and control over developing products as well as the required delivery of quality products within cost and schedule.

Additional activities designed to satisfy the goals of Phase II included a strong coupling with the DOD-STD-2167 Revision A activities, gaining insight into software quality metrics research and development and continued data collection as new data sources were defined. The coupling with the 2167 Revision A activities was necessary to produce a model which is compatable with and interfaces to the world of conventional software development.

The results of the Phase II modeling effort are documented in Volume II of this final report. The remainder of this report is outlined as follows:

- Section 2 contains a discussion of the methods used to gather all available data concerning the KBS software development process.

- Observations made through analysis of the data obtained are presented in Section 3. These observations pertain to both the conventional and KBS software approaches, including the difficulties inherent to both methods.
- A detailed evaluation of the case study data obtained is contained in Section 4. Common aspects and important features are highlighted.
- Section 5 presents a synopsis of KBS development methods versus DOD-STD-2167, conventional and KBS software interface issues as well as the application of KBS to SDI requirements.
- A list of references, bibliography, glossary, and a list of acronyms follow Section 5.
- Lastly, the report includes several appendices:
  - Appendix A details problems cited in the development of KBS and conventional software;
  - Appendix B presents the case study questionnaire; and
  - Appendix C presents summaries of the case study responses.

SECTION 2

Data Gathering Approach

2.1 Literature Search

The data gathering effort began with a comprehensive literature search extending to both on-line and hardcopy sources of information. The computerized DIALOG system[1] was used to access the following databases:

- INSPEC (Information Services in Physics, Electrotechnology, Computers and Control)
- ABI/Inform (American Business Information)
- The Computer Database
- NTIS (National Technical Information Service)

The DROLS (Defense RDT & E On-Line System) search service was also used to access the DTIC (Defense Technical Information Center) database.

Keywords pertinent to the study were used to identify articles from the various databases for further inspection. Numerous articles covering a variety of relevant topics such as artificial intelligence, expert systems, natural language processing and battle management were ordered as a result of the on-line search activity.

In addition to computerized search techniques, many hardcopy sources of information were reviewed such as textbooks, conference proceedings and journals. A complete annotation of all resources used in the preparation of this document is presented in the List of References and the Bibliography.

The literature search was an ongoing process since its inception. Namely, particular references listed at the back of an article or textbook under review were often ordered. This ripple effect enabled the study team to obtain and review an exhaustive set of resources corresponding to the subject study.

The other medium that was used for obtaining references and information about AI development issues was the ARPA/USENET groups for AI including both the moderated and unmoderated newsgroups. Numerous articles, technical reports and conference proceedings were ordered and reviewed as a result of references from the network discussions.

[1] DIALOG Information Services, Inc., a wholly owned subsidiary of Lockheed Corporation

2.2 Case Studies

2.2.1 Questionnaire

A comprehensive questionnaire was designed and used as the basis for conducting the AI/KBS system case studies. The questionnaire, contained in Appendix B, is divided into four parts:

1. Introduction
2. Background
3. Development Cycle, and
4. Miscellaneous.

The Introduction put the questionnaire in perspective by discussing the concept of a software development model. The Background section solicited general information about the subject system, such as purpose, the number of experienced engineers assigned to the project and the extent of success attributable to the system. The Development Cycle section contained the most questions and was subdivided into the following categories:

- General Procedures
- Requirements Definition/Analysis
- System Construction
- System Evaluation/Validation, and
- Field Support.

The overall purpose of the Development Cycle section was to obtain detailed technical and managerial information concerning the scope, development and acceptance of the system.

Lastly, the section titled Miscellaneous was used to capture data on lessons learned as well as allow the respondent to include pertinent information that had not already been requested.

Most of the responses received were in written form. In a few cases, oral responses were obtained using an interview or discussion forum with the respondent. In these cases, a written annotation of the discussion was generated and sent to the respondent to verify the accuracy of the reported information.

2.2.2 Selection Criteria for Cases Studied

Once the questionnaire was finalized, the primary goal was to obtain as many responses as possible. Maximizing the number of samples available for analysis enabled conclusions to be drawn which more closely reflected the general AI/KBS community. Distribution of the questionnaire was also

made to a wide variety of companies. In this wa , data was captured pertaining to different application areas and procedural methods for developing AI/KBS software.

The case selection process began by identifying two in-house KBS efforts. From that point, the literature search and review uncovered many companies and universities that are active in KBS software development. Because most of the university efforts are experimental in nature and without strict schedules for completing defined milestones, the tendency was to concentrate on the industrial sector. Companies that claimed to have built *successful* AI systems were of the most interest.

In addition to the literature search, data was obtained on-line from the Commerce Business Daily (CBD) files concerning AI awards since September 1982. Based on the descriptions of the projects awarded, candidates whose work seemed most pertinent to the subject study were selected. In addition, discussions were held with cognizant government agencies to ensure appropriateness of effort and verify claims in the literature.

As mentioned in the following section, attendance at various workshops/ seminars resulted in many contacts who were interested in responding to our questionnaire. Lastly, some of the contacts made resulted in references to other sources active in the KBS arena.

2.2.3 List of Participants

Based on the selection process, over 50 companies/agencies were contacted. Initial contacts were generally made by telephone to introduce the study and determine whether or not the source was interested in participating. In the majority of cases, the response to participate was positive. Although more than 40 questionnaires were sent out to interested parties, many respondents were prohibited from participating on the basis of proprietary information. Consequently, 26 completed questionnaire responses were received. The list of respondents is shown in Table 2.2.3-1 along with the name of the system to which the response applies. Case analyses are presented in Section 4 and summaries corresponding to the questionnaire responses for each participant are shown in Appendix C.

2.3 Other Methods

In addition to the literature search and case studies, project members attended several AI seminars to obtain information concerning the subject study. Individuals experienced in AI software development were consulted as needed to enhance the team's understanding of the discipline. Lastly, to gather data pertinent to the ongoing SDI architecture studies, a trip was made to the SDI Library in Falls Church, Virginia. These information gathering methods are described further in the succeeding paragraphs.

2.3.1 EIA Workshop

During the week of 16 through 20 September 1985, a team representative was present at an Electronic Industries Association (EIA) workshop entitled "Technology for the 1990's." This team

2.3.1 *EIA Workshop*

Table 2.2.3-1: List of Questionnaire Respondents

NAME	SYSTEM	APPLICATION
ARINC	STAMP	Equipment testability shell
BOEING	Strategic Force Management Decision Aid	Replanning decision aid
BOEING	ATR	Automatic Target Recognition
BRATTLE RESEARCH	Unnamed	Text interpretation
CARNEGIE GROUP	DISPATCHER	Factory monitor/control
DEC[a]	XCON	Computer system configuration
ETI	PEGASYS	Automatic pagination
FREY ASSOCIATES	THEMIS	Natural language processor
GTE DATA SERVICES	COMPASS	Fault diagnosis
IBM FSG	FDRS	Fault diagnosis
INFERENCE	Authorizer's Assistant	Charge authorizations
INFERENCE	Medchec	Fraud detection
LAS	ESP	Software costing
LAS	Frequency Hopper Signal Identifier	Signal identification
LGC	Pilot's Associate	Combat avionics assistant
MITRE (BEDFORD)	Liquid Oxygen Expert System	Fault diagnosis/detection
MITRE (McLEAN)	ANALYST	Tactical planning system
NORTHROP	ESTAS	Avionics decision aid
PGSC	CBTAO	Tactical cost decision aid
PGSC	DART	Target decision aid
PGSC	SPEA	Battle situation projections
SANDERS	TESS	Test equipment assistant
SCHLUMBERGER	Dipmeter Advisor System	Interpretation of logs
SA&E	DSDS	Decision support tool
SA&E	SFAS	Financial analysis system
TI	PRODUCTION SCHEDULER	Manufacturing production scheduler

[a] DEC is a registered trademark of the Digital Equipment Corporation

representative attended a tutorial on DOD Standard 2167 (hereafter referenced as 2167) and participated in a panel forum on *AI and its Impact on Current Software Engineering Practices*. In particular, the panel concentrated on the AI software development process and 2167. Many features of the AI development process were identified, most of which are neither addressed nor compatible with the most recent version of 2167. Specific areas of concern are discussed in 5.1 of this report.

In addition to the information obtained at the workshop, attendance was valuable because it led to several contacts who responded to the case study questionnaire.

2.3.2 Expert System Conference

In October 1985, a team member attended a two day *Expert Systems Conference* sponsored by the Data Processing Management Association (DPMA). There were nineteen speakers, three of whom were military personnel involved in AI applications work. Although the majority of the speakers were from the industrial sector, several were actually working on AI systems for the military. Pertinent topics covered included:

- managing the development of large expert systems;
- software engineering methods for expert system development;
- knowledge acquisition techniques;
- developing expert systems in Ada[2]; and
- a variety of expert system applications and potential uses.

The conference proved to be valuable in terms of obtaining information germane to the subject study and identifying potential respondents to our case study questionnaire.

2.3.3 TI Satellite Symposiums

Team members attended three Artificial Intelligence Satellite Symposiums sponsored by Texas Instruments, Inc. (TI). The dates and titles are as follows.

- November 13, 1985 *Knowledge-Based Systems And Their Applications*
- June 25, 1986 *Knowledge-Based Systems: A Step-By-Step Guide To Getting Started*
- April 8, 1987 *An AI Productivity Roundtable*

Each symposium featured renowned speakers in the AI field. For example, Dr. Edward Feigenbaum (Stanford University) participated in all three sessions.

Each of these symposiums included descriptions of several expert systems in routine use, and live question and answer sessions. Resource materials were also provided including a glossary of terms and an extensive bibliography.

[2] Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

2.3.4 Expert System Technology Transfer Seminar

From May 12-14, 1986, a team member attended an expert system seminar sponsored by Digital Equipment Corporation (DEC). Practical strategies for managing expert system development were covered from three perspectives: strategic/business, technical and human resource/organizational. The faculty included DEC personnel currently involved in expert system development projects. One of the more valuable facets of the seminar was a video taped case study, extending over the three day period. After each of the nine tapes were played, the attendees separated into groups to analyze salient issues and recommend a course of action.

The materials from the seminar include the *Guide to Expert Systems Management* manual which presents in detail DEC's ten step management procedure.

2.3.5 Consultation

Three individuals have been used as consulting resources since the inception of the project: Dr. Charles Rich of MIT, Mr. J.T. Ginn and Mr. David Harris, both of Sanders Associates.

Dr. Rich, an AI research scientist at MIT, has provided guidance concerning our solution strategies and reviewed selected materials generated by the project team. Dr. Rich is active in the field, having published numerous articles, given many AI related lectures and having received several grants from the National Science Foundation (NSF) and the Defense Advanced Research Projects Agency (DARPA).

Mr. Ginn and Mr. Harris, both active in the development of AI software at Sanders Associates, have provided continuous support in suggesting case study contacts, answering questions concerning the AI development process, reviewing all project documentation and making recommendations as appropriate.

2.3.6 Research at SDI Library

Two team members visited the SDI library in February 1986 to review SDI architecture study data as well as other materials pertinent to the study. Although architecture study information was unavailable, several articles pertaining to the study in general were identified and ordered.

2.3.7 Coupling with DOD-STD-2167

On two different occasions, team members met with the Joint Logistics Commanders (JLC) agencies concerning the Logicon, Inc. Revision A activities on 2167. The intent of this coupling was to provide input to Revision A concerning AI system developments and their applicability to the standard. Since the Revision A draft publication was completed prior to the KBS modeling activity, any influence on the proposed changes to 2167 was minimal. It is expected, however, that the planned release of Revision B will be a more appropriate time to closely couple KBS system acquisition with the development model of 2167 and to identify mismatches requiring resolution. Because of the nature of KBS versus conventional software development and a foreseen requirement to integrate

both into the same system (e.g., knowledge-based signal processing), it is important that continuous dialogue be maintained between KBS modeling activities and changes to the conventional software development models defined in 2167.

SECTION 3

Observations

3.1 Conventional Software Approach

3.1.1 Standard Waterfall Approach to Conventional SW Development

Large scale DOD computerized system developments, which began in earnest in the 1960's, took various approaches to the development of software. Although there were no formally defined models for software development within DOD, there evolved an accepted approach that became the pattern for software developments acquired under contract to the DOD. This approach documented by [6, Boehm], [42, Metzger], and others was characterized as the waterfall model for software development. Although the terminology and the break points between phases varied, the models essentially included the phases of:

- requirements definition;
- design,
- coding and debug;
- integration and testing; and
- operations and maintenance.

The Military Services, in the mid and late 1970's, began to recognize the waterfall model as a proper representation of a sound management approach and took actions to embody this model within the various policies and standards being developed at that point in time. For example, both Air Force Regulation (AFR) 800-14, written in 1975, and MIL-STD-1679(Navy), written in 1978 included some level of representation of the waterfall model. The 1679 standard went even further and included the engineering disciplines that were an outgrowth of the structured engineering revolution. The concepts of top-down design and structured programming were included as requirements for software being developed for the Navy.

Policies and procedures were developed to address the management and engineering problems known to exist during the mid-70's. As the application of computers within DOD systems expanded, problems with software continued to occur and the search to find ways to improve the software development process was an ongoing one.

The next step within DOD in the policy and software development standards arena came about as a result of a workshop sponsored by the Joint Logistics Commanders Computer Resources Management Group. The workshop, held in March 1979, put forth the recommendation that the

Services should and could pursue the development of software under a single policy and standard and that efficiencies could be achieved with a common approach within the DOD community.

As a result of the workshop, a working group was formed to develop a joint Service policy and standard for the development of software. The model chosen was a slight modification to previously mentioned waterfall models and was highlighted by its concentration on activities, products, reviews and baselines associated with the software development process. Both the policy and the standard, now known as DOD-STD-2167 and dated 5 June 1985, were developed in an effort to improve the DOD software development process with an eye towards eliminating the types of problems identified below. In addition to the standard, a set of data item descriptions (DIDs) and changes to Military Standards 483, 1521, and 490 were developed. This standards package represents the DOD approach to software development for the 1980's.

3.1.2 Reported Problems in Conventional Software Development

In preparation for the KBS modeling effort, it was necessary to review past problems in conventional software development so that past mistakes would not be repeated. Numerous studies had been conducted and significant effort expended to define and categorize these problems. With the knowledge of past development problems in hand, the modeling effort could be conducted in such a manner that the resulting KBS development model orientation would not repeat the past problems of the conventional software development world.

Many problems associated with conventional software surfaced during the development of DOD Defense Systems and can be categorized best under the following headings:

- Software Life Cycle;
- Software Environment;
- Software Product; and
- Software People.

The following sections examine the major problems in each of the stipulated categories. The problem categorization identified is based on a report issued by the DOD Joint Services Task Force, *Report of the DOD Task Force on Software Problems*. Appendix A presents a list of specific conventional and AI software development problems. [17, Druffel]

3.1.2.1 Software Life Cycle

Software Life Cycle refers to the development of conventional software from requirements definition and analysis to system maintenance and quality assurance. Software problems have been experienced in the following areas:

- Requirements;
- Management;

- Acquisition;
- Product Assurance; and
- Transition.

Requirements is the process that involves the analysis and definition of a system. In conventional software, some of the major difficulties with requirements surface from incomplete or inaccurate requirements documents and/or poor communication between users and engineers. In general, each of the stipulated problems can result in increased costs and unacceptable schedule delays. A system's success is particularly affected when the requirements are based primarily on available time and money.

Difficulties with management of conventional software development occur because of a variety of obstacles that relate to:

- insufficient budget allocation;
- unskilled management;
- lack of metrics, models and tools;
- undefined software acquisition methods; and
- determining how to develop software versus hardware.

The ripple effect of unskilled management is reflected throughout all phases of the software life cycle. For example, proper design depends highly on complete requirements which in turn rely heavily on good management communication. Another obstacle to good management results from budget estimates based on vague requirements or budgets determined from inaccurate models and metrics designed to determine software costs. Finally, without the necessary models, metrics and tools, management does not have the appropriate tools to adhere to software development standards requiring status reporting.

Acquisition of software and tools is not fully defined for conventional software development because government software acquisition has historically been based on the purchase of associated hardware. In the area of tools application, configurations of software are often not properly managed even though the tools exist to do so. Tools are often developed in-house and not made available to others for their use.

Testing is not provided at each life cycle phase because of insufficient funding and scheduling. In fact, uncertainties arise in determining the amount and kind of testing suitable to each life cycle phase. Testing problems occur most frequently where requirements are vague, incomplete or difficult to measure.

The transition of software, whether from exploratory research to engineering development, or from development to maintenance, introduces numerous problems that adversely affect conventional software. Two transition examples from which problems surfaced are: the introduction of microprocessors and firmware into software development and the distribution of management control (a need for a standard or policy has arisen?); and the transition of software from research and development to operational systems (how should it be accomplished?).

3.1.2 Reported Problems in Conventional Software Development

3.1.2.2 Environment

The Environment category deals with the tools and methodologies used in the development and support of computer software. Problems exist in the following areas:

- Disciplined Methods;
- Labor-intensive activities;
- Tools;
- Reinvention; and
- Capital Investment.

Conventional software needs to improve the use of engineering disciplines. A set of activities are required to develop and support software throughout the entire life cycle in order to create high quality software. Large development efforts are performed by many groups, either colocated or independently located. This demands disciplined control mechanisms if software development is to be properly managed.

As a labor-intensive technology, conventional software needs to concentrate on automating the manual process in order to improve the efficiency of people. For example, the manual process of reporting documents, status, etc. could be automated to allow professionals to concentrate on more difficult and important tasks.

The lack of standardized tools has resulted in problems with acquisition and software development. A large number of tools tend to be inaccessible, difficult to use or inefficient. Software tool acquisition could become more definable and therefore more achievable with a standard set of tools that provide consistent computer support, particularly for software development. For example, high order languages should strive for machine independence that would improve transportability.

Software reusability is considered by many to be a powerful means of reducing software reinvention. However, attempts to reuse software inappropriate to the given application may actually hinder the development process and degrade the resultant product.

Capital investment is needed to improve the support environments and reduce problems such as reinvention or the lack of adequate methodologies and tools. There are three main problems associated with the lack of capital for conventional software expenditures. First, software is developed on out-of-date hardware not designed to support the software effort. Second, schedule slips due to the lack of adequate tools introduce higher overall costs. Third, not enough capital is invested in IR & D projects.

3.1.2.3 Software Product

The Software Product category is defined as the operational computer software and materials necessary to provide life cycle support. These include requirement and design specifications, source code, test data, system generation data, unique support tools, etc. Problems have been experienced in the following areas:

3.1.2 Reported Problems in Conventional Software Development

- Doesn't Meet the Need;
- Software Metrics;
- Design Attributes;
- Documentation; and
- Immutable Software.

'Doesn't meet the need' refers to the users dissatisfaction with the system. An inferior product can easily be developed as a result of unclear, vague or incorrect requirements. System performance is highly dependent on the type and quality of testing. In both situations, users end up with a system which simply does not work to perform the mission.

Software metrics provide essential analytic models and empirical data on software to help in the selection of software engineering techniques for estimating development resources and to evaluating future costs. However, few effective models were found in use to validate conventional life cycle costs and productivity during the development and support phase. Few good analytic models and methods are available to gather empirical data to estimate future costs and system impacts. No method to determine how to develop firmware has been established. Lastly, managers do not receive the status reports necessary for cost and schedule analysis.

Design attributes deal with the provision of an acceptable program solution to problems defined in the requirements specifications. Some of the major areas where design problems have been identified are: not adhering to good software development techniques; inadequately designed requirements; and understanding software versus hardware implications. Improperly designed software is often the result of not following a defined software methodology, a lack of adherence to a top-down hierarchical system breakdown and/or a lack of consideration to human engineering as an important design issue. Another major design problem occurs when requirements are vague or incorrect and the engineer makes the wrong assumptions to implement the requirement. Also, correct software design decisions depend on a basic understanding of the hardware involved. Since conventional software normally cannot handle system modifications without added cost expenditures or schedule delays, ineffective system design severely hinders the system's success.

A major problem affecting conventional software documentation is that financial resources allocated to its preparation do not necessarily reflect actual costs. Therefore, whenever a schedule slips, or the budget is overspent, the documentation effort is relaxed. Consequently, successive software changes are not reflected in the documentation. Another problem stemming from a reduction in documentation is the determination of what needs to be documented. For instance, system interfaces must be documented and the documentation of requirements and design is important. Also, traceability between documents is critical but generally not possible because of unavailable tools.

One of the problems with the development of conventional computer software is that it is generally tailored to the specific application or hardware environment. Consequently, software packages are often system unique, non-portable and non-reusable (Immutable Software). Acquisition agencies are then forced to repeatedly pay for software which could have been available elsewhere.

3.1.2.4 People

Qualified software personnel for managerial and technical positions are required to avoid project problems. People problems exist in the following subcategories:

- Skills;
- Availability; and
- Incentive.

The rapid spread of digital technology has resulted in a widespread shortage of skilled system engineers, software engineers and managers. Consequently, the three main problems of professional skills, professional availability and professional incentive have been identified. A highly valued and respected managerial skill is the ability to guide software through the life cycle from requirements to maintenance. However, the skills necessary to do this requires a broad range of software experience and acquired knowledge. Few educational programs exist to provide professionals with these necessary skills. Therefore, the lack of available, experienced and skilled software professionals continues to exist. Professional incentives are needed to attract and retain expert software professionals. A lack of reward for excellence and the competitive software market in search of skilled software professionals results in high turnover rates.

3.2 AI Software Approach

AI techniques have typically been oriented towards ill-structured problems with solutions that are characterized as heuristic, non-algorithmic and non-deterministic. AI technology application areas that reflect these characteristics include robotics, vision, natural language, automatic programming, planning and expert systems. The AI solution approach to these areas is in contrast to conventional software methods that generally deal with well-defined algorithmic problems and deterministic solutions.

The characteristics associated with AI applications have had a strong influence on the approaches taken to AI software development. Uncertainty of tasks, knowledge, results and functionality have prompted AI programs to be written to gain a better understanding of the problem domain. Because of this uncertainty, rapid changes in the solution approach are expected as more is learned about the problem domain. This has led to the development of powerful tools and integrated environments for the development of AI software. The result is that AI development approaches are iterative in nature and accomplished by small development teams using highly integrated development environments.

The AI technology environment that has supported the non-algorithmic, heuristic approach to problem solving includes programming languages, development methodologies, and development environments. These technology areas have been prime contributors in supporting small development teams in their quest to handle large problem domains.

AI programming languages inherently support development under uncertainty. The longest surviving AI language is LISP. It supports delayed committment and language extensions as two approaches to managing uncertainty.

Development methodologies are closely tied to the iterative problem solving approach to AI software. Both the iteration cycle and the need to delay decisions or specifications until development of a mature prototype is complete are inconsistent with conventional software development methodologies. AI software development methodologies typically begin in the middle of a problem domain and spiral out to a complete, acceptable solution rather than adopting a top-down hierarchical decomposition approach typical of conventional software methodologies.

AI software development environments reflect a committment to personal computing. These environments absorb some of the tedious and routine work such as: garbage collection; memory allocation; integrated editing, debugging and inspection; and documentation. They also allow handling of knowledge based facilities. Background processes in AI development environments provide tracking information about programs, display presentation of the information, and provide active agents to recognize conditions, propose action and perform cleanup. LISP machines and AI shells are examples of such development environments.

3.3 Generic Description of the AI Development Process

AI system development has been described as a highly iterative process and generally resembles a *build a little, test a little* approach with numerous feedback loops. Close examination of the AI development process reveals a set of activities that are integral to the development of an AI system. These activities, which are a synopsis of the models to follow, include:

- Problem Definition;
- System Design;
- Implementation and Testing; and
- Support.

Coupling these activities with the concept of building system increments allows for the delivery of partial capabilities for use early in the development process. Experience through use can then be fed back into the development process so updates can be made to future increments prior to delivery for field use.

Several of the better documented methodologies are described in the following paragraphs. These methodologies recognize the iterative, incremental approach to AI system development. The generic descriptions differ somewhat in definition and phasing, however, they are all generally supportive of the previously identified activities that are applied in an incremental manner.

3.3.1 The Expert System Model

In *Building Expert Systems* [35, Hayes Roth] the authors define the major stages of knowledge acquisition for expert system construction as:

- Identification;

3.3.2 The DEC Model

- Conceptualization;
- Formalization;
- Implementation; and
- Testing.

Although defined in terms of knowledge acquisition, the stages also represent how an expert system is constructed. Each stage defines a set of activities related to expert system development. As the authors point out, these stages are not sequential in nature but instead overlap each other. Further, development of an expert system does not constitute a single pass through the stages. There is continuous feedback from each of the stages to all preceeding stages thus representing the incremental nature of the KBS knowledge acquisition process.

Further examination of the stages of knowledge acquisition reveals a mapping between stages and the previously defined phases of Problem Definition, System Design, Implementation and Testing, and Support. Identification equals Problem Definition and involves defining the problem domain, developing informal descriptions of the problem and partitioning it into subproblems. Conceptualization and Formalization parallel the System Design phase activities of knowledge acquisition for a selected subset of the problem domain and developing a partial specification detailing the tools and representations thought to be appropriate for the problem domain. Implementation and Testing includes the actual construction of the KBS system or its subset in the chosen language environment followed by a test of its correctness.

The five stages are further mapped into two phases rather than the four activities identified above. The two phases are:

- Phase 1 - Identification and Conceptualization
- Phase 2 - Formalization, Implementation and Testing

The differences in phasing do not represent a conflict but instead offer two different views.

3.3.2 The DEC Model

In *The Artifical Intelligence Experience: An Introduction* [59, Scown] , the author defines a model for expert system development which essentially includes four phases. These phases are:

- Problem Identification;
- Functional and Design Specification;
- Create Bounded Prototype, and
- Incremental Development

Problem Identification includes the definition of the problem domain. Functional and Design Specification equates to the previously generic phase of System Design wherein a subset of the knowledge domain is defined and a functional specification for a bounded prototype is developed. In addition, a design specification is developed detailing the tools and representations thought to be appropriate to the problem domain. Creation of the bounded prototype represents Implementation and Testing for the first deliverable product. Incremental Development represents all subsequent development and testing activites associated with deliveries on an incremental basis in the evolutionary, iterative environment of AI system development.

The DEC model closely tracks the generic model previously defined. The development model views the process as a set of sequential activities to the point of achieving a bounded prototype. At this point it becomes the iterative process of develop an increment, test it and then release it to the field for further feedback and support.

3.3.3 The Dipmeter Advisor

The Dipmeter Advisor expert system experiences reflect two different aspects of phasing considerations [62, Smith]. One reflects an oscillation that occurred during the project and the other the phased approach taken to develop successive prototypes for the Dipmeter Advisor.

The oscillation is best explained by examining the two phased development process that occurred in respect to tool development and prototype implementation. The first phase constituted a feasibility demonstration in which knowledge was collected for a constrained problem and the tools selected appropriate to the domain. The second phase included an expanded implementation of the domain knowledge where the expert system development tools remained relatively constant. As development progressed, points in time were reached where the initially defined tools did not allow further expansion of system expertise. The development process was interrupted at this point while the phase one activity was restarted to construct a more powerful set of tools.

The Dipmeter Advisor project went through a three phase development process of:

- Feasibility Demonstration
- Utility Performance Demonstration
- Utility/Performance Evaluation

The Feasibility phase included the activities of knowledge acquisition and prototype implementation, and as such really could be conceived of as two separate phases. The Utility Performance Demonstration phase examined the product for its ability to provide the correct answers but also examined the viability of the prototype as a commercial product. The Utility Performance Evaluation phase consisted of field evaluation of the prototype. All of these phases included problem and enhancement feedback for preparation of the next prototype.

3.3.4 Other Generic Models

There are numerous other models which have been developed which in some way parallel one of the previous mentioned models [29.22, Harmon, Gates]. These models reflect phasing which is both overlapping and iterative in nature, and an approach that relies on extensive prototyping.

3.3.5 Generic Definition of KBS Developments

Defining a generic description of the KBS development process is a task that requires the identification of the steps or phases in the process and associating activities with each phase. The iterative nature of KBS developments further complicates any visualization of the process and requires that one think about it as a repeating process. The generic phases of Problem Identification, System Design, Implementation and Testing, and Support, as summarized in Table 3.3.5-1, represent a phased pattern that has been largely substantiated by the available literature and one that was used as a starting point for the modeling effort.

Table 3.3.5-1: Generic Model Phases

Model vs Phases	Expert System Model	DEC Model	Dipmeter Advisor Model	Harmon and King Model
Problem Definition System Design	Identification Conceptualization Formalization	Problem Identification Functional and Design Specification	Knowledge Acquisition	Problem Selection Prototype Development and System Design
Implementation and Testing	Implementation Testing	Bounded Prototype and Incremental Development	Prototype Implementation and Demonstration of Utility and Performance	Delivery of Computer System and System Evaluation
Support		Delivery	Evaluation of Utility and Performance	Integration and Maintenance

The activities associated with each phase have been widely documented in the literature and accepted by organizations with significant expert systems development activities. The chosen generic description provides the foundation for the discussions and data represented in the remainder of the report.

3.4 Description of KBS Development Characteristics

There are many characteristics associated with a KBS development project. Most of these characteristics either have no counterpart or are exercised differently when compared to conventional software development activities. In this subsection, detailed descriptions are provided for the following KBS features:

1. Knowledge acquisition;
2. Knowledge representation;
3. Reasoning methods;
4. Redundancy exploitation;
5. Development environment;
6. Exploratory programming style;
7. Rapid prototyping;
8. Small development teams;
9. User involvement;
10. Documentation produced;
11. Testing; and
12. Management control mechanisms.

The manner in which the above features relate to conventional software development characteristics should be evident from the discussion that follows.

3.4.1 Knowledge Acquisition

A major aspect of the development of expert systems resides in knowledge acquisition. The process involves the transfer and transformation of problem-solving expertise from a knowledge source such as human experts, data bases, and literature, to a program. There is no such activity in the algorithmic world of conventional software development because knowledge is not explicitly represented as a separate system component. Unfortunately, knowledge acquisition represents the main bottleneck in expert system development because of the difficulty with extracting knowledge for incorporation into a knowledge base. As a result, system development time is prolonged. Another problem is based on the fact that only a limited number of automated tools exist to aid in the acquisition of knowledge from the domain expert. Therefore, knowledge acquisition depends on the skill of the knowledge engineer who must effectively communicate with the domain expert in order to understand and structure the knowledge. Knowledge acquisition might involve several

planned interviews between the knowledge engineer(s) and the domain expert(s) with the purpose of extracting, summarizing and structuring the knowledge.

Knowledge acquisition begins with an identification of the expert system's participants, goals and problems. Generally, one knowledge engineer is assigned to research the system's knowledge domain and hold interviews. Likewise, one domain expert usually represents the expert system needs. However, depending on the scale and complexity of the solution space, more than one engineer or expert may be assigned to a project. The knowledge engineer's responsibilities to research the domain and schedule meetings with the expert is designed to aid in the process of identifying the system's problems and goals.

An inherent difficulty with knowledge acquisition lies in helping the expert to structure his/her knowledge, to identify and to formalize domain concepts. Domain experts cannot always logically express the manner in which they solve problems. Therefore, a number of interviewing techniques are available to the engineer to help measure performance and uncover expertise. Audio tapes of the interviews are not uncommon.

As key problem-solving methods are identified, a means to produce and refine the expert's knowledge generally follows. Two well-known methods are to either transcribe the problem-solving knowledge onto paper or to produce a prototype. The motivation behind both methods revolves around a way to present the expert with a visual aid which allows him/her to mentally step through the process to verify the validity of the problem-solving technique. Essentially, gathering an expert's knowledge involves an incremental approach to define and redefine problems until the domain knowledge contains an adequate amount of information to respond appropriately to a problem.

The main idea which drives the knowledge acquisition process from identification to conceptualization of knowledge, revolves around the need to formalize the concepts, problems and information flow into a more formal knowledge representation which defines the data structures, inference rules and control strategies. A recognized dilemma with knowledge acquisition resides in identifying an expert whose knowledge adequately reflects the problem domain. Apparently, an incremental approach to software acquisition, in the form of several prototype releases, resolves the conflict of whether data gathered is appropriate to the system's needs. The approach allows the domain expert an opportunity to watch the system work which enables him/her to comment on erroneous system problem-solving. In turn, the knowledge engineer is provided with a tool with which to identify problems areas such as: what kind of knowledge is not sufficiently defined; what kind of logic needs to be redefined; and what new areas of knowledge should be acquired.

Although the availability of tools to aid in the knowledge acquisition process is very limited, some tools do exist. (See Section 3.4.5 for a list and explanation of these tools). For example, there are tools that help construct and refine expert systems. Although the tools do not acquire rules, they sometimes allow the expert to define and organize the rule building blocks.

3.4.2 Knowledge Representation

Knowledge acquisition and knowledge representation are closely intertwined, nonsequential processes. Throughout the development period of a knowledge based system, these steps are revisited many times to initially define and then refine the knowledge base. In designing a knowledge based system, it is not an easy task to determine how the knowledge can best be represented to guide

the system's problem solving behavior. Nonetheless, in order to extract knowledge from an expert, it may be helpful to understand apriori how the knowledge will be represented. However, without already having the knowledge, it's not even clear what has to be represented. Consequently, in typical systems, a relatively small but central fraction of the domain expertise is collected by the knowledge engineer interacting with the expert over a short period of time (i.e. one to two months). During this period, the knowledge engineer has a chance to evaluate the knowledge, observing patterns and levels of abstractions before implementation decisions are made.

How knowledge is structured in a program is highly dependent on the conceptual framework: whether or not knowledge is centered around objects or processes; or thought of symbolically. Since a well-defined knowledge representation methodology can simplify complex problems, understanding knowledge in order to choose from a large variety of available techniques becomes vital. The following subsections examine four commonly referenced techniques to represent knowledge: semantic networks, frame-based, rule-based and logic programming methods.

3.4.2.1 Semantic Net

Semantic net, a knowledge representation method based on a network structure, consists of points called nodes that are connected by links referred to as arcs. The links define the relationship between the different nodes. Each node in the semantic net generally represents physical or conceptual concepts, events or objects. A variety of choices exist for the definition of arcs, depending on the knowledge represented. Some of the more familiar arcs used to represent the hierarchical relationship between nodes are *is-a* and *has-part*. Natural languages tend to use arcs such as *agent*, *object* and *recipient*. As a simple example from the statements: 'human is a mammal' and 'man is a human', one can infer that man is a kind of mammal if the semantic net represents the knowledge as a hierarchical breakdown where man is a subset of human and human is a subset of mammal.

Some common mechanisms or features of semantic nets are inheritance and demons. Inheritance is the ability of one node to adopt properties from nodes at a higher level on a hierarchy. A property inheritance, which can be implied from an is-a relationship, means that instances of a class can have all characteristics of classes to which they belong. Thus, man is an instance of human and inherits properties of human which is a kind of mammal. Naturally, redundant information, and therefore wasted storage is avoided. Demons are another aspect of semantic nets. The main focus of demons is to provide a resource of information which can be used as values whenever needed or specifically requested of the database.

3.4.2.2 Frames

Frames are another method of representing knowledge (facts and relationships). Similar to semantic nets, frame-based knowledge representations make use of a network of nodes connected by relations into a hierarchy. Frames differ from semantic nets in that nodes are defined by groups of attributes (also referred to as slots). In turn, each slot may contain attribute values, default values, pointers to other frames, sets of rules, procedural attachments (that execute whenever attribute values are referenced), etc. Basically, top-level nodes in the hierarchy represent general concepts and lower nodes refer to specific instances of concepts and therefore inherit properties of higher-level nodes.

3.4.2 *Knowledge Representation*

The procedural attachment allows for two complementary ways to state and store facts: a procedural or declarative representations. Declarative representations are explicit because they are simply an assertion about a fact. In contrast, procedural representations are more difficult to define because facts are specified by how they are used.

The power of frames consists in an ability to efficiently combine declarative with procedural representations. Because of their modularity, declarative representations are more easily maintained and adaptable to independent and changing facts. Procedural attachments are more efficient, but more difficult to maintain

3.4.2.3 Rules

Rules are another form of knowledge representation, based on IF *condition* THEN *action* statements. Basically, when the facts stipulated in the IF part of the rule are true, the THEN action part of the rule is executed. When this happens, the rule is considered *fired* or *executed*. Results may be manifested in an instruction for the system to add a new hypothesis to the database, program control, or peripheral device control.

The pairing of the IF portions of rules to facts result in *inference chains*. The inference chain shows how the system uses rules to reach its conclusions.

There are basically two *inference mechanisms* (ways in which rules can be set up in a rule-based system): forward chaining and backward chaining. Forward chaining refers to the gathering of pieces of information in an attempt to build forward to an end goal. Backward chaining begins with a goal and works backward to seek a chain of premises that accounts for all the facts at hand.

Rules, as a knowledge representation scheme, provide a natural environment for depicting processes driven by rapidly changing, complex environments. Rules not only provide specifications on how a program should react in relation to changing data without necessarily knowing the flow of control, but can also easily explain what a program did or how a conclusion was reached.

3.4.2.4 Logic Programming

Logic, a fourth type of knowledge representation, encompasses a variety of logical notations and is based on the model of proof in mathematical logic. Two common notations are propositional logic and predicate calculus.

Propositional logic deals with statements that are either true or false. Each statement can be linked together by a connective such as *and, or, not, implies* and *equivalent* which result in a compound statement. Rules exist to propogate the truthfulness of compounds. Other rules support inferencing

Predicate calculus is considered an extension to propositional logic. Elementary units are referred to as objects and statements about objects are considered predicates. Therefore, the statement is son(Bruce,Tom) is a two-place predicate with two objects, Bruce and Tom. The predicate can be evaluated as a true or false assertion that Tom is or is not the son of Bruce. Predicates can be linked into larger expressions by means of the same connectives used in propositional logic.

PROLOG, which stands for PROgramming language for LOGic, is a classic example of a language which incorporates some of the principles of predicate logic. The PROLOG control structure is logical inference. Basically, one states facts and rules about objects and PROLOG can determine whether a specific conclusion can be deduced given the collected data.

3.4.3 Reasoning Methods

There are numerous reasoning methods reported in the literature. Consequently, the discussion in this section is generally limited to the more common strategies and those methods reported on in the case studies presented in Section 4.

Three areas of reasoning that might guide a KBS system through its knowledge base include:

- *inference strategies*;
- inexact reasoning; and
- control.

Inference strategies are generally based on the use of logical axioms. The most common inference strategy used in knowledge systems today is the application of a logical rule called *modus ponens*. [29, Harmon] The method states that if the premise(s) of a rule is true, then the conclusion(s) is true. Namely, when A is known to be true and an axiom "If A, then B" exists, it is valid to conclude that B is true. The application of modus ponens has two implications. First, the method is simple so that reasoning based upon it is easily understood. Secondly, not all valid conclusions can be drawn with the use of modus ponens alone. *Modus tollens* can be used to enrich the inferencing strategy. This logical axiom states that if B is known to be false and an axiom "If A, then B" exists, then it is valid to conclude that A is false.

Resolution is another, more general inference rule. The basis of resolution is that if there are two axioms of the form:

- A or B
- not B or C

then A or C logically follows. The expression A or C is called the resolvent of the first two expressions. Resolution can be generalized so that there can be any number of disjuncts in either of the two initial expressions as long as one expression has a disjunct that is the negation of a disjunct in the other expression. In the general case, conclusions drawn by logical axioms such as modus ponens and modus tollens can also be arrived at using resolution.

Because a knowledge base may not necessarily contain all the information required to reach a conclusion, inferencing techniques must include reasoning about uncertainty. In other words, like an expert, the inference engine must be able to deal with incomplete information. Inexact reasoning has been implemented using either numerical or non-numerical techniques.

3.4.3 *Reasoning Methods*

Numerical methods may include the use of confidence levels or certainty factors attached to a particular conclusion and propogated as further inferences are made. For instance, *if a man is wearing a white collar, he must be a priest* may be associated with a 30% confidence level.

Another numerical approach to dealing with uncertainty is the use of Bayesian decision theory. Simply stated, this method associates a subjective probability value with every hypothesis to measure the degree of belief. In the absence of evidence, any hypothesis is assumed to have an initial probability which changes as evidence is gathered. The ultimate goal is to choose the hypothesis with the highest probability.

The fundamental concern with numerical methods of handling uncertainty is that they hide the reasoning that produces them. [14, Cohen]. In addition, while numbers are easy to propogate over inferences, what the numbers mean may not be clear. The *theory of endorsements* is a non-numerical method of inexact reasoning. Because inexact reasoning is a knowledge intensive task, the theory of endorsements employs a heuristic approach to dealing with uncertainty. Endorsements are records of information that affect one's certainty, including the kind of evidence available and the methods used to produce the current hypothesis from uncertain preconditions. Endorsements are propogated over inferences using heuristics in a manner that is sensitive to the context of the inference. Furthermore, endorsements can be ranked in an effort to choose the hypothesis with a superior level of endorsement.

Endorsement is similiar to recording justifications in a *truth maintenance system* (TMS) [14, Cohen]. The crucial difference is that in a TMS, a justification is used to decide whether a conclusion has support, but the kind of support (or evidence) is irrelevant.

Both endorsements and TMS support nonmonotonic reasoning wherein conclusions that are true at one instance may need to be retracted. Nonmonotonic systems generally include a dependency network approach to logic which maintains dependencies between values. Truth values are propogated using constraints supplied by logical expressions. With dependency networks, one can keep track of justifications made for all conclusions which can then be modified if previous assumptions are withdrawn. For example, in a planning system, it may make sense to proceed in a certain way. As more information becomes available, an earlier decision may need to be retracted. As a consequence, all the implications based on that initial decision also need to be retracted.

In a monotonic reasoning system, all values concluded remain true throughout the course of the program run. In other words, facts that become true remain true.

In addition to inference strategies and inexact reasoning, there are two primary problems associated with a knowledge based system that are addressed by the control portion of the inference engine:

- where to begin the reasoning process; and
- what to do when alternative lines of reasoning emerge.

The first problem can be answered by the use of *forward* and/or *backward chaining* mentioned in Section 3.4.2.3. If the possible outcomes are known and if they are reasonably small in number, then a backward chaining or goal directed approach is very efficient. On the other hand, if the number of possible outcomes is large or if the possible goal states are not known at the outset, a forward chaining or data driven approach is needed.

Planning islands represent another approach to the first problem. Namely, begin processing where there is the most information and the least amount of uncertainty.

The second problem can be addressed by deciding on either a *depth-first* or *breadth-first* search. With the more common method, depth-first, the inference engine focuses on searching for detail and descending to deeper levels to produce a subgoal. A breadth-first search sweeps across all possible premises in a rule before digging for greater detail.

In addition to depth-first and breadth-first, there are many *more* search strategies that have been employed in KBS system developments. Nonetheless, a discussion of additional strategies is beyond the intent of the discussion herein.

With any of the control strategies discussed above, the inference engine can provide the basis for an explanation facility by keeping track of where it went to form any particular conclusion.

3.4.4 Redundancy Exploitation

One of the features of KBS development that has impact on performance, reliability and quality is the redundancy of information and processing. This characteristic is in part due to the nature of the more common knowledge representation schemes and inference mechanisms and in part to the "middle-out" problem solution approach. Typically, each item is evaluated or tested several times before all conditions and changes are satisfied. KBS systems are generally built recursively so a module may be used or executed for a variety of purposes. Use of both forward and backward chaining in the same system provides redundant methods of inference. Support of multiple lines of reasoning is common in the more complex expert systems. All of these factors work to provide a higher reliability measure for the code providing the right answer since it has been tested with a variety of inputs in a variety of contexts. Similarly, the data or knowledge has been massaged by several different processes so unexpected results become less common.

Redundant knowledge and inference processing work together to reduce the effects of uncertainty and missing data. During the development process, they help to highlight inconsistency. With proper development team discipline during the period of iterative changes to the KBS system, it becomes robust in handling uncertainty and inconsistency. Issues of trustworthiness make this aspect of redundancy exploitation more important in the case of autonomous systems than ones with more human interaction and control. Strategies for its employment in *BM/C*^{3} applications hinge on the continuation of research to define conditions for this redundancy and incorporation of this knowledge in a set of knowledge based tools used by the development teams.

3.4.5 Development Environment

Development environments for KBS software have been among the most powerful available because of their emphasis on a user interface, debug facilities and tight coupling to the language of choice. Efforts in the last twenty years to create a programming environment and tools for the production of large and complex programs have been successful in providing high resolution graphics, multi-windowing and an integrated environment to enhance user productivity by shortening the cycle of edit, compile, link, debug, and execute. Increased productivity due to this cycle shortening has

been accomplished through a reduction of operational steps in the cycle, and the capability to do incremental compiles and function evaluation.

Tools developed include the programming languages themselves such as LISP, PROLOG and OPS5. LISP has the capability to create and embed languages which have been used for tasks such as dependency analysis, browsing, inference, truth maintenance, constraint propagation and knowledge representation. A class of computers called LISP machines provides the power of an integrated software development environment by integrating these tools with editors, menuing systems, and the capabilities of the LISP language. There are some hardware enhancements that support this environment including high resolution graphics display, data type tagging, and directed machine architectures. Future capabilities will include parallel processing and memory features which will allow more powerful search and control strategies to be executed within the machine resources.

Special tools exist to aid in the generation of expert systems. These tools, called Expert System Shells, are available to assist in the development of many standard architectures for expert systems. These shells provide additional integrated environment services. Inference engines, knowledge representation schemes and graphics oriented interfaces provided by the shells are readily available for installation on a wide variety of host machines.

The commercial proliferation of expert system shells and the wide availability of differing tools and languages provides many choices in the selection of an environment. Important considerations other than cost include the amount of overhead that is allowed, suitability of the representation of knowledge, and inference techniques that are required. There have been many observations of an iterative cycle of tool building and knowledge acquisition, which may be shortened by the appropriate selection of tools if enough is known about the problem area initially.

There are efforts underway to provide further power in the integrated environment for the use of development teams. Other efforts of value to the KBS developers and project personnel include the development of a knowledge based editor of which KBEmacs is one of the most widely documented examples. Tools to aid in the development of KBS software, but to be used by personnel other than the AI development engineers, would include a test management assistant and a project management assistant designed to import knowledge from the development area and aid the test team or the management team in reaching decisions affecting the development and test of the system.

Increased capability in the development environment to obtain the best performance from the computer language, machine technology and development team will provide benefits to the BM/C^3 development effort in schedule, cost and performance risk reduction. Of primary import to the success of KBS projects that can have impact on the BM/C^3 problem is the need to identify and develop tools that can be used to support multiple teams of developers. This class of tools would also include some of the tracking of progress necessary for the management and test assistants.

3.4.6 Exploratory Programming Style

Exploratory programming is the "conscious intertwining of system design and implementation" [61, Shell]. Recognizing that some applications are design, rather than straight implementation problems, the notion of exploratory programming allows the design to surface from experimenting with the program. In essence, the design and the computer program are developed together.

This technique is extremely valuable when dealing with large and complex systems (e.g. *BM C*^{3}) for which it is extremely difficult to postulate a complete specification. Some reasons behind this difficulty are:

- Complexity itself - a design engineer simply cannot anticipate all of the requirements of a large scale system in advance of the implementation.
- Fluid requirements - for instance, during development, the hardware changes or the particular databases for which the system is to consult changes.
- Human factors - it's difficult to specify user interface requirements in advance. Interfaces generally undergo extensive empirical testing to determine whether or not they are effective and considerable redesign to make them so.

Regardless of the reason, a large system with changing specifications leads to disaster when conventional software development procedures are used. Namely, the existing technology assumes that the specification is fixed and that the implementation conforms to the specification. However, if a major change in the specification arises, every phase of the project may have to be redone - preliminary design, detailed design, coding, unit test and integration. Of course, the later the specification changes in the development cycle, the more work that has to be redone. In addition to compromising the budget and schedule, the quality of the end product is also likely to be affected in the rush to complete the system as close to the deadline as possible.

With exploratory programming, the specification is expected to change as a result of the implementation. For example, suppose an initial system specification is generated. Once the system begins to be implemented, some of the uncertainties may be better understood, or perhaps concerns will be uncovered that had not been previously identified. Consequently, the specification is modified and the exploratory programming effort is redirected. This iterative process continues until at some point, portions of the system requirements begin to stabilize. However, it is important to recognize that exploratory programming may not be an end in itself. Because of the many revisions made, the code may be inefficient and unstructured when it first achieves functional acceptability. If efficiency and structure are important issues, the code can be reimplemented using traditional top-down techniques. By the time this stage is reached, the specification is not likely to change in a major way and the implementation is not expected to be done more than once.

Because exploratory programming calls for quick implementation of a system, and is generally performed by a small development team, computer systems conducive to this activity must enhance the programmer's productivity. Namely, programming tools such as those discussed in the previous subsection must be available to the development team.

3.4.7 Rapid Prototyping

Rapid prototyping involves a process of recycling through system implementation and testing in order to determine the validity of a knowledge representation scheme. Essentially, the prototype, a product of this process, represents an aspect(s) of the expert system and acts as a tool to demonstrate in what manner encoded facts, relationships and inference strategies reflect the expert's

3.4.7 *Rapid Prototyping*

knowledge. This section outlines the cyclic nature of rapid prototyping and successive refinement in relationship to AI system development.

The rapid prototyping process begins with acquisition, conceptualization and the formalization of knowledge into specific data structures, inference rules and control strategies. The next step involves the implementation of a demonstration prototype in order to test the adequacy of the formalization. The prototype's knowledge base is implemented through the aid of tools such as intelligent editors, languages etc. which, if not available, are developed. Perhaps the most important aspect of the prototype lies in the selected knowledge representation scheme.

The demonstration prototype's purpose resides in proving the feasibility of the system in order to obtain a financial commitment from the user and in testing certain aspects of problem definition, scoping and representation of the domain. Therefore, heavy user involvement is necessary to the financial success of the system and a strong domain expert commitment is required to determine the system's accuracy. In fact, the expert usually works along with the knowledge engineer to discuss problem-solving strategies throughout all the development phases. Ultimately, the expert recognizes and points out ineffective problem-solving techniques in the prototype.

However, the success of the prototype also depends on the ability of the knowledge engineer to formulate a set of performance criterion with well-defined input and output for test measurement purposes. The idea is to facilitate an expert's ability to recognize the prototype's efficiency and accuracy particularly with regard to the knowledge base and the inference structure. One knowledge engineering technique includes the use of predetermined case studies which are initially reviewed in an interview with the domain expert and then compared with the prototype's means of handling the same information. The expert must be aware of the prototype's purpose to properly ascertain its functions. Testing becomes more complex when one considers that a prototype only reflects a fraction of the system's actual capabilities. Frequently, a prototype sacrifices performance in order to concentrate on functionality or concentrates on the user-interface and sets aside system functionality. The expert's involvement to test the prototype usually encourages more determination to gather the appropriate knowledge and alerts the expert on how the system functions in order to know how to fine tune the actual product.

A successive refinement of the prototype generally follows the demonstration. The prototype has served a purpose to identify errors such as incorrect inference rules, invalid knowledge and inaccurate control strategies. Another cycle through implementation and testing corrects the errors and brings up further problematic areas. Prototype development from the initial demonstration to the mature system seems to involve a five step process: [67, Waterman]

- demonstration prototype;
- research prototype;
- field prototype;
- production model; and
- commercial system.

The demonstration prototype, as previously described, attempts to prove the feasibility of a system for financial support purposes and to test the knowledge representation and problem-definition techniques. The research prototype is generally a medium-sized program with powerful performance capabilities, but with a tendency to suffer from insufficient testing. The field prototype, an average to large-sized program also displays good performance. However, the field prototype differs from the research prototype in that the system is much more reliable because of extensive testing. The production prototype tends to perform well, be reliable and fast. Oftentimes, the production prototype has been reimplemented in a more efficient language. Finally, few systems actually reach the stage of a mature system which is a production model used commercially.

3.4.8 Small Development Teams

Although KBS systems constructed to date are large, they have typically been developed by *small* teams. In 1982, a survey was conducted with the intent of estimating the number of manyears required to build various AI systems [15, Davis]. In the survey, ten systems were represented, including the well known MACSYMA, HEARSAY, DENDRAL, MYCIN, PUFF, PROSPECTOR and XCON systems. One interesting result of the survey is that every system included was developed by an average full-time manpower effort of two to five people.

In conventional software systems where the code is generally modular and functions are loosely coupled, the project can logically support a large staff. Namely, one or more people can be assigned to design, code and test each major module. This concept cannot be extended to KBS systems where development does not follow top-down, modular techniques.

The development of expert systems requires people with a variety of skills. The following breakdown is typical for a moderately difficult problem domain [67, Waterman]:

- Domain expert (75% commitment)
- Senior knowledge engineer (25% commitment)
- Junior knowledge engineer (100% commitment)
- Programming support (100% commitment)

The total effort for the above listing is three full-time professionals. Waterman extends this concept to address more complex systems. The summary shown below depicts the size of the development team as a function of project complexity:

- Moderately difficult : 2-4 people.
- Difficult : 3-5 people.
- Very difficult : 4-6 people.

In all cases, it is crucial that a company insure the accessibility and availability of the domain expert(s). The expert(s) must devote up to 75% of their time in the beginning stages of the project and up to half-time thereafter to ensure the possibility of a successful project.

3.4.9 *User Involvement*

The knowledge engineers on the project extract information from the domain expert(s) and encode this information into the chosen method of knowledge representation. The knowledge engineers may actually construct the underlying framework in which to encode the domain knowledge or purchase an off-the-shelf development tool to do so (Section 3.4.5)

Programming support can be used to aid the KBS and/or conventional programming effort, depending on the project needs. Conventional programming may be needed to implement the more algorithmic portions of the system, or to integrate the system with existing software such as databases or sensor inputs.

Small development teams constructing large systems imply high productivity where team members often work at or above their ability to manage system complexities. Nonetheless, the development process is strongly influenced by the availability of powerful software development tools (Section 3.4.5) that help to manage this complexity.

The small development team concept also seems to preclude the need for frequent documentation since information can be verbally disseminated with relative ease. Namely, changes made to a program can be verbally communicated to team members eliminating the need to generate documentation such as Engineering Change Orders (ECO's), revisions to previous system documentation or even internal memos.

The majority of the case study evaluations shown in Section 4 indicate development teams comprised of less than 6 people. This observation is generally consistent with the literature. On the other extreme, where system complexity and magnitude are very high, the size of the development team may be much larger. For example, the team supporting the ongoing production of XCON, a DEC computer configuration system, numbers 35. The project team is broken up into four major factions:

- Administrative support (3);
- User support (5);
- Technical support (conventional software) (6); and
- Knowledge engineering (21)

More details pertaining to the XCON system are presented in Section 4 and Appendix C. The point is that the scale and complexity of SDI BM/C^3 systems may be comparable to XCON. If this is the case, construction of systems in the SDI context may deviate from the concept of a small development team effort.

3.4.9 User Involvement

A user may be simply defined as anyone who uses the KBS system, including the domain expert(s), knowledge engineer(s), tool builder(s) or actual end-users. The tool builders may be involved in debugging the system in later stages, whereas the knowledge engineer uses the tools available to implement and refine the system knowledge. An end-user can be thought of as the person(s) for whom the system was developed. Although essential, the importance of end-user involvement in the KBS system development process may not be obvious. Consequently, it is useful to consider the stages of system development in which end-users can positively influence project evolution:

- **System definition** - Similar to a conventional software project, it is essential that the system meet the user's needs. Consequently, user input must be considered in the system specification phase.

- **Prototyping** - As discussed in Section 3.4.7, prototyping allows the users to observe and comment on the evolving system. A particularly important feature of any system is the user interface, which often comprises a large segment of the total system. For example, in the Dipmeter Advisor project, the following breakdown in percentage of code was observed [62, Smith]:

  - Inference Engine 8%
  - Knowledge Base 22%
  - Feature Detection 13%
  - Support Environment 15%
  - User Interface 42%

  The amount of effort expended on developing a user interface for the Dipmeter Advisor is consistent with other projects whose intentions were to facilitate running of the system by non-sophisticated users [63,58, Stanley, Schwartz]. Furthermore, if the system is cumbersome to use or if the output, such as results or error messages are not clear and relevant, users are not apt to abandon existing manual methods.

  Explanation facilities, whereby a system explains how solutions were reached, are also a pertinent part of a user interface. A user gains more confidence in the system when he becomes comfortable with the reasoning process employed to reach a conclusion(s). The explanation data may also be useful to the tool builder and knowledge engineer in debugging the system.

- **Testing** - End-users and the domain expert often play an important role in testing by hypothesizing test cases and/or reviewing results.

The majority of the case study evaluations presented in Section 4 indicate that the user community was actively involved in system development and that user satisfaction was often used as a measure of system success. If the users are an integral part of the development process, they're less likely to feel threatened by the system or experience the "not invented here" syndrome. The result is a group of users who look forward to system completion in light of utilizing the end product to facilitate their jobs.

The encouragement of user input throughout a KBS system development process differs from that of a conventional software engineering project whereby a predefined number of reviews are held at fixed points (Section 3.1.1). Perhaps increased user participation in the conventional world would lead to fewer misunderstandings concerning system functionality and a higher success rate in terms of customer acceptance.

3.4.10 Documentation Produced

The absence of an agreed upon model or method for developing KBS systems has resulted in the lack of any standard related to the types and kinds of documented information necessary for the

3.4.10 Documentation Produced

development and support of KBS systems. In *Building Expert Systems*[35, Hayes-Roth], the author discusses problem identification during the Identification stage, defining key concepts and relations during the Conceptual phase and developing partial specifications during the Formalization phase. All of these events have reference to information that could well be captured in the form of a document. For instance, there are numerous references to providing users documentation on how to run a KBS system. However, there are no details concerning what level of detail and what kind of information should be included in the documentation for the system [35, Hayes-Roth].

Other document requirements in the development of KBS systems are discussed in which the development of a paper knowledge base, a knowledge acquisition grammar, internal knowledge base formats, knowledge base, inference engines, and interface are all presented as project documents [20, Freiling]. The documentation scheme presents information that results from a six step process defined as:

- familiarization;
- organizing knowledge;
- representing knowledge;
- acquiring knowledge;
- inference strategy design, and
- interface design.

On the XCON project at DEC, the functional specifications were understood rather than written because of the ongoing revision process that occurred during the development [59, Scown]. Design specifications were embodied in the coded product since expert systems do not lend themselves to defined algorithmic solutions where specifications can be written.

The need for documentation on KBS projects has, to some degree, been driven by the nature of KBS developments that have occurred to date. First, most KBS projects are small with 2 to 5 developers involved. Because of the limited number of development team members, little need for documentation exists. The iterative nature of KBS software development efforts also make it very difficult to determine what and when to document. It is clear that a problem definition needs to be agreed to but that implementation should not await complete definition. The introduction of an early prototype leads projects to produce a working model for the customer so that feedback and proof of concept can be demonstrated before a large commitment to the project is made. The *produce something quickly* approach allows little documentation time because prototyping is oriented towards a product approach rather than a process approach (as defined in the conventional software development world).

Another factor affecting the need for documentation is the tools available to support the development of KBS systems. Some of the environments available provide a level of sophistication not generally seen in conventional software developments. These environments include various graphical and windowing tools which allow users to do development in an environment which does not rely upon extensive paper documentation.

Finally, the research and development nature of KBS projects put them in a category where technology feasibility is of the essence and not the ultimate deployment and support of the product. In these cases, emphasis is certainly not on documentation since management insight takes on a different flavor and users are not expected to enter the picture until some indeterminate future point in time.

3.4.11 Testing

Testing is the comparison of a system's *actual* behavior with its *intended* behavior.

In conventional software development, a developer is typically presented with a system specification which is a statement of the problem to be solved by the new system and some idea of what a solution to that problem might be. Implicit in such a specification is a functional statement of the systems behavior under a number of real or envisioned situations. The developer takes that specification and embarks on the software development process involving several steps:

- *Requirements Definition* a first level decomposition of the problem and envisioned solution and an allocation of requirements from the system specification to the highest level components of the system.
- *High Level Design*.
- *Detailed Design*.
- *Coding and Debugging* (or unit testing) in which program code is developed and structural, as opposed to behavioral, errors are detected and removed.
- *Integration*.

Following the integration step, a completed but unproven software system exists. This system should exhibit the developer's interpretation of the system behavior defined in the specification. There is no assurance, of course, that it is the problem solution suggested by the specification.

In parallel with the development team, the test team translates the customer specification into test procedures that model the system behavior in terms of stimulus/response pairs (as in "Push the button; the light turns green" or "Enter the file name; the system responds 'no such file'." This process involves

- *Requirements Definition* in which the specification is examined for essential requirements whether explicity stated, implied, or derived during the design process.
- *Test Specification*.
- *Procedure Generation*.

3.4.11 *Testing*

Because the conventional software model usually requires that the test procedures be approved by the customer prior to the initiation of any formal validation/testing, the procedures are considered a correct interpretation of the specification. The test process consists of comparing the software as built with this "right answer."

The ability to specify functional behavior, as is done during conventional software development, implies that the problem to be solved is well enough understood so that executional details of the resulting solution are known. Specifications do exist in the KBS world, but they tend to be problem statements without much of a hint as to what an acceptable solution might be.

As they proceed from the problem statement, KBS software developers often enlist the aid of "experts" to guide them in the direction of an acceptable solution. This permits the generation of a system which displays some approximation of this expert's idea of solution behavior. Because they desire independence from the development team and because the number of experts is limited (frequently to a single individual), a similar approach in the preparation of test procedures is available to the test team. Indeed, even if a different expert is available to assist in the development of the test process, the potential for disagreement with the other expert exists. The question then becomes one of determining the "right answer." In this case, unlike that of conventional software development, the customer is a doubtful arbitor because of the fuzziness of the original problem.

Even presuming that some approach toward generation of agreed-to test procedures can be found, additional problems plague those charged with testing the KBS system:

- Does the system *correctly* reflect the knowledge given it?
- Is the knowledge given the system *correct*? Who decides? Systems can be envisioned where this may not matter, at least not in the same way as conventional software, since the product may be self-correcting[1]. In this case, the question may become "When the system discovers that it is incorrect, does it move toward correctness by an acceptable amount?" Again, who decides?

Just as AI software is urged towards solution behavior by prototyping, the testing of AI software for battle management should be prototyped. Examples where this could be applied include DARPA Stategic Computing Programs for AIRLAND 2000 and the NAVY FRESH program. Additionally, upgrades to the PATRIOT and AEGIS systems incorporate AI technology. Experiments with their test programs could be undertaken.

Development of a *test advisor* knowledge based system would enable the test community to analyze the test requirements for systems and the results of tests more efficiently. This would allow fuller test team participation from project inception and give the benefit of a learning curve that included the prototyping phase of system development. Such an innovation might then permit combining the fairly common "build a little, test a little" AI philosophy with a new "test a little, build a little." This latter, implying examination of evolving requirements and partial systems, is as much a driver of the system result as the evolving system is of the test process. There is a synergy in this effort to define goals and criteria for test, tolerance and performance. As both test and development teams identify and commit to these, the problem to be solved becomes less certain. This strategy would correlate well with a spiral approach aimed at minimizing system development risks.

[1] A review of both current literature and Sander's survey results indicates that self-correcting software is not a common occurence.

3.4.12 Management Control Mechanisms

KBS projects are typically accomplished by small development teams and therefore do not require the extensive management control mechanisms that are usually necessary for large scale DOD system programs. Most AI research and development projects are developed by a single team with very little interface necessary to other organizations or systems. Their control usually consists of periodic technology reviews of actual versus planned progress. The review process is generally informal with no predetermination of what information will be discussed. This is in contrast to the typical large scale DOD software development program which has planned management and product reviews at various stages during the development process. These reviews are often extensive requiring a large resource committment.

The management techniques employed for KBS systems is complicated by the iterative nature of KBS development. Mastery of the traditional software development model has hardly been attained and KBS systems enter the scene to present a whole new problem set for modeling. It is clear that few managers even understand the world of AI let alone know how to manage the highly complex iterative process. Considering the difficulty of scoping a domain within an expert system and the definition and building of small increments, it is difficult to conceive of a means for managing and controlling a total KBS development project from requirements to product deployment and retirement. For that reason most KBS developments are actually level-of-effort developments. That is, only near-term schedule, cost and performance objectives are defined. Progress is then measured in the near-term and not over a long span of time. Since there are no proven software cost models for estimating KBS software cost and schedules, predicting the future is even more complicated. The lack of a broad historical data base of KBS software costs and schedule also offers no solution or aid in predicting costs and schedules.

Within the AI system application area, there have been few systems that have been developed under the constraint of schedules and costs normally associated with DOD systems. KBS acquisitions, which call for the development of systems for delivery to the DOD within a given timeframe, are just beginning to appear within DOD. These procurements are normally of a standalone expert system which has little or no interface with other systems. It is also noteworthy that these systems are not real-time to the same degree as an embedded Battle Management Command and Control System must be.

As KBS system applications grow in size, there will be an increased need to divide the problem solution into manageable modules in order to develop AI systems under dictated schedules and costs. It has been suggested that benefits be derived in the maintainability, testability and speed areas by applying modularization [47, Orciuch]. Management of large projects can potentially benefit from a modularization approach by providing meaningful subsets of a system on which a given team can work on. One of the experiences from a project the size of XCON is that bringing someone new onto a large project can take a significant amount of time to become familiar with the design. Modularization should allow individuals to become productive sooner.

The key to managing a KBS project or program is adequate insight and visibility into the development process in order to assess the status and to predict the future of the program at any given point in time. The evolutionary nature of the KBS development process only further complicates the management job. As KBS applications continue to expand and increase in size, management will be faced with further challenges in managing and controlling the development process.

Table 3.5.1-2: Artificial Intelligence Software Problems

ARTIFICIAL INTELLIGENCE SOFTWARE PROBLEMS

	Problem Statement
1.	The rapid prototyping method does not offer clear guidance on how to produce a well-engineered commercial product
2.	Little knowledge exists on how to manage the transfer of a progressive and evolutionary technology
3.	There only exists a limited number of engineers in AI software development.
4.	There is a need for multiple, specialized representatives in the field of AI.
5.	The knowledge is often ill-specified because the expert cannot always express exactly what he/she knows about the domain.
6.	With rapid prototyping, the system is always in a state of flux.
7.	There exists a lack of knowledge regarding testing of AI systems.
8.	There often exists voluminous amounts of information in the definition of an expert system.

3.5 Difficulties in Developing AI Systems

The following section is concerned with identifying AI software problems cited from a literature study undertaken at Sanders Associates, Inc. and the case study responses received from the Sanders AI Questionnaire. A comparison of AI and conventional software development problems is made. The literature study examined major problems that appeared most frequently in different AI software subcategories which are defined and outlined in Appendix A. The case study problems deal with those stipulated in one or more responses. The comparison of AI and conventional software development problems compares and contrasts common software problems and issues.

3.5.1 AI Software Development Problems Cited in the Literature

Certain problems in AI software development were cited in more literature sources than others. Table 3.5.1-2 lists the eight problems which appeared with the most frequency. (See Appendix A for a complete examination of all the software development problems identified in the literature sources). The remainder of this section examines each of the frequently identified problems in more detail.

AI software development is a relatively young field in comparison with conventional software developments. Therefore, difficulties with identifying and determining the best means to develop AI systems has arisen. The problem is compounded by the fact that one of the most widely accepted methods for AI software development resides in the use of rapid prototyping which offers no clear guidelines on how to produce a well-engineered commercial product.

The lack of well-defined software standards and methodologies filters into the proper management of AI software development. AI managers are confronted with a progressive and evolutionary technology which by definition entails little guidance on how to properly manage people and the effort.

A by-product of a new technology can be found in the limited number of experienced personnel. AI software development is hurt by both the lack of available and/or experienced engineers or managers. In order to develop standards and methodologies, AI needs specialized personnel to determine an appropriate set of standards to effectively manage and guide an AI software effort.

Knowledgeable engineers could help improve the knowledge acquisition process, the main bottleneck in AI software development, by providing effective techniques to communicate with the domain expert. The process of extracting knowledge for incorporation into a knowledge base is time-consuming and difficult. Adequate gathering of information depends on the knowledge engineer's skill at extracting information from the domain expert who oftentimes has difficulty expressing what he/she knows about the domain.

The success of an AI system depends on the ability of the knowledge engineer to formulate a set of performance criterion with well-defined inputs and outputs for test measurement purposes. With rapid prototyping, the idea is to improve an expert's ability to recognize a prototype's accuracy particularly with respect to the knowledge base and the inference structure. Since the prototype only reflects a small fraction of the system's capabilities, and uncertainty is a major factor in the selection of a solution approach, testing becomes more complex. Also the constantly changing requirements introduced by a system which is in a constant state of flux, contribute to the inability to effectively test an AI system. Once again, the process involved in testing AI systems and the lack of knowledge regarding testing of AI systems seriously affects the AI software development effort.

The last major AI problem identified concerns the need to delimit the task of a problem when building an expert system. Knowledge-intensive problems may not be clearly bounded. Unless the abundance of problem-solving information is bounded, the role the expert system will play may not be sufficiently defined to produce a well-defined and effective system.

3.5.2 AI Software Problems Observed from the Case Studies

Many common AI software development problems can be identified between the case studies gathered from the AI questionnaire (see Appendix B). The most frequently mentioned problems were:

- a lack of experience in managing expert systems,
- a need to free knowledge engineers from tasks which take away from creative work,
- a lack of domain expert commitment which is vital to system success, and

3.5.2 AI Software Problems Observed from the Case Studies

- the need for better testing methods.

However, a variety of other significant problems also surfaced with regard to:

- the need to develop a prototype;
- the ability for a system to support the views of various experts;
- the need to allow for requirement changes;
- the need for continual financial support throughout the AI system development effort;
- documentation; and
- overhead introduced in real-time systems by shells.

An examination of these problems follows.

AI lacks the required historical background from which an appropriate guide on how to develop a well-engineered product can be extrapolated. Consequently, management does not have the guidelines or the experience necessary to help in the administrative process of AI software development. For example, the system's success depends on management's ability to effectively allocate and manage domain experts and knowledge engineers. The need for a domain expert's time commitment cannot be over emphasized. Appropriate knowledge acquisition, effective problem-solving techniques and schedules are highly dependent on the expert's knowledge and availability. Without a committed effort by the domain expert, schedules are delayed and further expenses are incurred. Another problematic area occurs when knowledge engineers are not provided with management's support to spend the necessary time and effort to design the most appropriate knowledge representation and inference mechanism. Some suggestions have been to encourage the knowledge engineer's creative mind during the prototype phase by freeing him/her from administrative duties.

Another problem area for management involves adequate testing and maintenance of the AI system. Unfortunately, AI software tends to be more difficult to test because of the complexity involved in demonstrating the feasibility of concepts with regard to requirements. Generally, because of the inexact nature of AI systems, multiple or even thousands of test results could be the correct responses to one requirement, and thousands more may exist. Consequently, the need for a more appropriate means to test AI software has been recognized. With regard to maintenance, the need to modularize the system has arisen for larger systems to ease the maintenance task.

Other problems associated with AI software development concern user commitment to the AI project, system modifications to accomodate multiple expert ideas, the overhead introduced by shells, and standards needed for documentation. The user's financial commitment to an AI software development effort is highly dependent on the development and demonstration of a prototype. Lack of a user's commitment to a specification and architecture for systems with embedded AI software has been identified as a problem. The need to allow several experts to examine the prototype oftentimes results in a modified or adapted user-interface. A third problem involves the overhead introduced by shells in real-time systems. And finally, a discrepancy exists on how to document, what to include in documents, and whether documentation is even necessary.

A general concensus among the responses obtained from the case studies is that problems were significantly reduced by the flexibility that rapid prototyping and success refinement provides. Changes to the requirements, a natural and expected occurance with AI software development, were facilitated through this process.

3.5.3 A Comparison of AI and Conventional Software Development Problems

Many similarities and differences can be found in a comparision of conventional and AI software development problems. The following section attempts to compare and contrast some of the most apparent problems. Four major software categories, based on a report issued by the Department of Defense (DOD) Joint Services Task Force on Software Problems, *Report of the DOD Task Force on Software Problems*, [17, Druffel] shall be examined : Software Life Cycle, Environment, Software Product and People.

A brief summary of problems common to both AI and conventional software development follows:

- ineffective communication between the users of the system and managers;
- lack of skilled software development managers and engineers;
- methodologies are in a state of transition;
- acquisition of tools is not fully defined;
- difficulty determining amount of testing appropriate to each life cycle phase;
- lack of adequate Quality Assurance engineering practices;
- software is a labor-intensive technology;
- software is reinvented;
- user satisfaction;
- lack of well-defined metrics;
- shortage of skilled system engineers, software engineers and managers; and
- problems with users who do not have enough time to spend on the project.

Each of these problems, which are discussed in subsequent paragraphs within the four software categories, has a significant impact on the software development process.

3.5.3 A Comparison of AI and Conventional Software Development Problems

3.5.3.1 Software Life Cycle

The Software Life Cycle refers to the relationship between different phases of the software development process from initial requirements definition and analysis to system deployment and maintenance. The main components of the Life Cycle category are Requirements, Management, Acquisition, Product Assurance and Transition. Interestingly, many of the problems that surface at the requirements phase, such as cost and schedule, documentation, communication and changes to the software are also characteristic of other life cycle phases, in particular management and acquisition. A brief comparison between conventional and AI life cycle problems follows.

As previously indicated, some Requirement problems involve changes in software, cost and schedule, documentation, and communication. The affect that requirements changes have on system development due to vague, incorrect, or missing requirements differs between conventional and AI software. In conventional software development, requirements are frozen at some predetermined point and any changes involve a formal process which may or may not permit the change. In fact, a minor requirements change could result not only in a large software effort to implement the new requirement, but could also introduce additional expenditures and schedule delays. These cost incurriments are even more significant when one considers that oftentimes requirements are defined according to available time and money. In contrast, Artificial Intelligence systems are developed with the underlying assumption that requirement changes will be incorporated as new information is gathered from prototyping. AI systems generally are developed in incremental stages of defining requirements, developing a prototype, and redefining new requirements based on the additional information obtained through research or from the prototype. Therefore, requirements can be expected to evolve. In fact, a major goal of *AI systems is to encourage and expect the system to* change.

Effective communication between the users of a system and engineers is another major requirements problem. In general, an engineer's comprehension of a user's needs is reflected in the ability to adequately document the information. Although conventional and AI software are similar with respect to communication problems, AI is affected differently by lack of communication. For conventional software, an understanding of the product being developed is essential to the system's success. In contrast, the basis for many AI systems, in particular Expert Systems, is an expert's knowledge. Therefore, the knowledge engineer has to communicate well with the user as well as understand the expertise.

Management of software encompasses various aspects of engineering: managing of software, people, and skills; availability of tools, metrics and models. A characteristic of both conventional and AI software development teams is the lack of skilled project managers. AI development is particularly affected because managers with or without AI development experience have no clear guides on how to produce a well-engineered product.

In both conventional and AI software development, the acquisition of software and tools is not fully defined. One problem is that conventional software developments are not always carefully tracked. With prototyping, AI systems are always in a state of flux which makes configuration management difficult. Tools are not necessarily a required deliverable, nor are they budgeted for in conventional software. In AI, tools are not necessarily identified before implementation.

Similar problems in Product Assurance can be found in conventional and AI software development. Difficulties arise in determining the amount of testing appropriate to each life cycle phase and

defining an adequate means to test requirements. However, unlike AI systems, conventional software can generally be tested except where requirements are vague or incomplete. In contrast, oftentimes no definite method to check whether AI requirement conditions are met exists.

Presently, conventional and AI Software Methodologies are in a state of transition. Conventional software is concerned with upholding standards and conventions for requirements, design and coding, while AI is in a state of defining a methodology.

Overall, many common problems which deal with *communication, requirements modifications*, tools, and software development are characteristic of both conventional and AI software. On the other hand, each of the problem categories vary based on the software application. Where communication with a user is vital to conventional software's success because requirements are frozen at some point, AI software works around the communication problem by developing a prototype which serves to further understand and properly implement the requirements. As a final example, test methods and approaches have always been a debatable subject in conventional software. Likewise, AI software suffers from the same problem. However, AI differs in that unlike conventional software where some adequate test method is usually available, a means to efficiently test AI software does not necessarily exist.

3.5.3.2 Environment

Environment encompasses tools and methodologies involved in the development and support of computer software. The five main categories described are: *Disciplined Methods, Labor-intensive*, Tools, Reinvention and Capital Investment.

A characteristic problem of conventional and AI software is the lack of adequate quality assurance engineering practices. Conventional software suffers from the need for improved disciplined methods to measure software quality particularly when requirements are not well-defined. AI software differs in that software quality tends to be difficult to test or measure at all. In fact, no true scale available can determine how high a rating an AI system can achieve.

Another environmental problem, the labor-intensiveness of software technology, is apparent in conventional and AI software. A need to automate manual processes with an ultimate goal of minimizing manual processes has been recognized. A recommended means of significantly reducing an AI manual process is to automate the process of acquisition, organization and structuring of knowledge.

Reinvention, a problem in conventional software, refers to the inability to reuse functionally similar software developed for other systems resulting in higher development costs. AI software, built in incremental stages which introduce small extensions, relies on the concepts of reusable software within a system. However, reusable software between functionally similar systems is questionable.

Sufficient capital investment in conventional and AI software could significantly reduce environment problems such as reinvention and lack of or inadequate methodologies and tools. No general recognition of the importance of capital investment to improve support environments and thus reduce problematic areas exists for conventional software. AI software suffers from a general lack of funding for AI system development.

3.5.3 *A Comparison of AI and Conventional Software Development Problems*

3.5.3.3 Software Product

Software Product deals with the operational embedded computer software and the materials necessary for life cycle support such as: requirement and design specifications, source code, test data, system generation data, unique support tools, etc. This section compares and contrasts major problems between conventional and AI software for the following categories: Doesn't Meet the Need; Software Metrics; and Design Attributes. Presently, no common problems between two other categories, Documentation and Immutable Software, have surfaced.

A problem with software occurs when the system does not meet user needs. Conventional and AI software depend on user satisfaction for a system's successful deployment. However, each differ in the manner in which users must be satisfied. Vague or incorrect requirements in conventional software seriously affect the final product. Oftentimes, errors in requirements result in an inadequate system which does not meet with user specifications. An interesting contrast is AI software, which developed incrementally, encourages and enforces regular meetings with the user to discuss and update system requirements. Therefore, requirement errors are not a serious hindrance to the system's deployment. However, AI software appears to confront a problem with the user's initial definition of the system. Frequently, a user's concept of the capabilities of an AI system exceed present day system development possibilities. Conventional and AI software do have one common user satisfaction problem. Both lack effective methods to determine the system's quality and therefore do not have a means to estimate or predict a user's level of satisfaction with the final product.

Software metrics attempt to provide analytic models and empirical data on software to help with the selection of software engineering techniques, to estimate development resources and evaluate future costs. A problem with conventional and, in particular, AI software is the lack of well-defined metrics. Poorly defined metrics result from the fact that various approaches to conventional software exist and no specific AI methodology is available. Given the variety (or lack) of standard engineering methodologies, standard metrics are difficult to define.

System design should provide an acceptable programming solution to problems identified in the requirements document. Some of the major problems in conventional software design are inadequately designed requirements, incorrect assumptions made by the engineer, and ambiguous requirements. In each of the situations, the system's ability to be modified becomes crucial. Some of the reasons for poorly designed software can be attributed to lack of a design methodology with a top-down hierarchical breakdown of the system and lack of consideration for human engineering in the design of the system. The result can be a system that is not necessarily capable of handling changes easily and with minimal cost expenditures and/or schedule delays. The final product essentially will not meet with user requirements.

Unlike conventional software, no formal methodology exists by which AI systems can be developed. No clear guidelines on how to efficiently produce a well-engineered commercial product through rapid prototyping is available. Nonetheless, many of the decided problems in conventional software design, which cannot easily handle requirement modifications for poorly designed systems, do not appear (or are remedied) in AI systems. As a result of the cyclic nature of AI systems, where software is evolved repeatedly from requirements to design to redefinition of requirements, incorrect system specifications are incorporated and updated continuously. In fact, perhaps the main problem

with the design of AI systems, when considering user needs, is in defining the knowledge base with optimal problem solving techniques.

3.5.3.4 People

In conventional and AI software development, two problematic areas have arisen as a result of a shortage of skilled system engineers, software engineers and managers: the number and availability of skilled professionals. Engineers with knowledge in various computer-related fields as well as experts who can successfully lead software projects are needed for conventional and AI software development efforts. As a relatively young field, AI cannot provide a sufficient number of experienced managers and engineers. Strong management does not exist to guide an AI project through completion and there are not enough engineers with AI software development experience.

Another conventional and AI software problem is a user's skill and availability to effectively communicate a system's requirements. AI, in particular, is hurt by users who trim knowledge to fit the knowledge structure; who are not able to express their knowledge; and who do not have enough time to devote to the project.

SECTION 4

Case Study Results

4.1 Case Study Data

Tabular data pertaining to the case study evaluations are presented in this section for the following participants:

1. ARINC Research Corporation;
2. Boeing Computer Services;
3. Boeing Military Airplane Company;
4. Brattle Research Corporation;
5. Carnegie Group, Inc.;
6. Digital Equipment Corporation;
7. Expert Technologies, Inc.;
8. Frey Associates, Inc.;
9. GTE Data Services;
10. IBM Federal Systems Group;
11. Inference Corporation (2 cases);
12. Lockheed Aircraft Service Company (2 cases);
13. Lockheed-Georgia Company;
14. The MITRE Corporation (Bedford, Ma.);
15. The MITRE Corporation (McLean, Va.);
16. Northrop Avionics Division;
17. PAR Government Systems Corporation (3 cases);
18. Sanders Associates, Inc;
19. Schlumberger-Doll Research (literature source only);
20. Software Architecture and Engineering, Inc. (2 cases); and

4.1 Case Study Data

21. Texas Instruments, Inc.

The tables present a common set of features with individual applications for each participant. The manner in which the data is codified facilitated the analytical process of ferreting out common characteristics, general trends and distinguishing traits. Specific observations relating to such aspects are presented in Section 4.2.

Prior to evaluating the tabular data,the reader is encouraged to compare the cases in a qualitative manner since:

- Many systems are in different phases of development. Consequently, the responses for a system in an early stage of development may change as the system matures. For example, a system that is in the phase of demonstrating technology may have had little user involvement. However, once the system has demonstrated feasibility of the technology, users are apt to become more involved.
- Many of the systems relate to different domains and applications. The features pertaining to the development of a tool many be justifiably distinct from the features relating to a complete stand-alone system.
- The concise responses required by a tabular presentation of the data may, in some cases, be misleading. To obtain a better understanding of the systems studied, the reader is encouraged to review the case study summaries delineated in Appendix C.

Table 4.1-1 : ARINC Research Corporation - System Testability and Maintenance Program (STAMP)

FEATURE	APPLICATION
Problem Category	Generic tool
Domain	Electronic warfare testability
Current System Phase	Field support
Type of Funding	Internal Research & Development
Experienced AI Staff	2
Size of Development Team	4-8
Level of User Involvement	Very active
Domain Expert Role	Not applicable
Formal Development Procedures	Yes, internal standards
Type of Knowledge Representation	Rules and facts
Inference Mechanism	Information theory and dependency analysis algorithms
Requirements Analysis	Yes, for the rehosted version
Approval Mechanisms	Both managerial and technical reviews
Iterative Development Process	Yes, 5 major architectural changes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	6 months
Development Time: End Product	2 years field model
Documentation	Yes, extensive
Configuration Management	Not after initial rehosted system
Conventional SW Interface	STAMP is written in conventional software (Fortran 77)
Tools Used: Requirements Analysis	No
Tools Used: Development	Fortran 77 compiler/debugger, and HP-1000 operating system
Tools Used: Testing	No
Formal Test Procedures	Testing: module, integration testing. 2 month user trial period
Testing Criteria	Not applicable
Test Data	Actual usage
Self Modifying Code	No

Table 4.1-2 : Boeing Computer Services - Strategic Force Management Decision Aid

FEATURE	APPLICATION
Problem Category	Replanning decision aid
Domain	Employment of strategic forces
Current System Phase	Development
Type of Funding	Internal
Experienced AI Staff	Yes
Size of Development Team	1
Level of User Involvement	High
Domain Expert Role	Knowledge acquisition
Formal Development Procedures	No
Type of Knowledge Representation	Frames, logic & rule based
Inference Mechanism	KEE supplied
Requirements Analysis	Yes
Approval Mechanisms	None
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	6 Man-months
Development Time: End Product	System not fielded
Documentation	Some - informal
Configuration Management	None
Conventional SW Interface	None
Tools Used: Requirements Analysis	None
Tools Used: Development	KEE, TI Explorer
Tools Used: Testing	None
Formal Test Procedures	None
Testing Criteria	Expert acceptance of plan
Test Data	Test scenarios
Self Modifying Code	No

Table 4.1-3 : Boeing Military Airplane Company - Automatic Target Recognition (ATR) Program

FEATURE	APPLICATION
Problem Category	Interpretation - image understanding
Domain	Target recognition
Current System Phase	Feasibility demonstration
Type of Funding	Internal Research & Development
Experienced AI Staff	Yes
Size of Development Team	Proprietary
Level of User Involvement	Influenced & directed development efforts
Domain Expert Role	Minimal expert consultation
Formal Development Procedures	Yes
Type of Knowledge Representation	Frame-based with rules
Inference Mechanism	Forward chaining
Requirements Analysis	Yes
Approval Mechanisms	Continuous internal review process
Iterative Development Process	Yes
Design Changes	Yes, minor rule changes and major user interface modifications
Prototypes Built	Yes
Development Time: Initial Prototype	Proprietary
Development Time: End Product	Proprietary
Documentation	Mostly informal except for annual report on state of research
Configuration Management	Entity checkpointing and file backup
Conventional SW Interface	Knowledge Craft supports direct calls to Lisp code
Tools Used: Requirements Analysis	No
Tools Used: Development	Knowledge Craft on Symbolics processors
Tools Used: Testing	No
Formal Test Procedures	No
Testing Criteria	Results verified by user and consistent with requirements
Test Data	Test image inputs - also fired every rule
Self Modifying Code	Not at this time

Table 4.1-4 : Brattle Research Corporation - Text Interpretation System

FEATURE	APPLICATION
Problem Category	Text interpretation
Domain	Business information
Current System Phase	Development
Type of Funding	Venture and contract
Experienced AI Staff	Yes - all members
Size of Development Team	12 overall
Level of User Involvement	Slight - more later
Domain Expert Role	Not applicable
Formal Development Procedures	Yes, internal
Type of Knowledge Representation	Proprietary (frame-like)
Inference Mechanism	Inheritance & deduction
Requirements Analysis	Yes
Approval Mechanisms	Yes - both management & technical
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	1 year
Development Time: End Product	2 years (estimate)
Documentation	Yes - formal specs & design documents
Configuration Management	Yes
Conventional SW Interface	Yes - in development
Tools Used: Requirements Analysis	Prototyping
Tools Used: Development	Symbolics environment
Tools Used: Testing	Yes - built own tools
Formal Test Procedures	Blind & regression
Testing Criteria	Absolute accuracy
Test Data	Text articles
Self Modifying Code	No

Table 4.1-5 : Carnegie Group, Inc. - DISPATCHER Project

FEATURE	APPLICATION
Problem Category	Monitor and control
Domain	Factory automated materials handling
Current System Phase	Completed - production model
Type of funding	Commercial contract
Experienced AI Staff	2
Size of Development Team	3
Level of User Involvement	Present but passive
Domain Expert Role	No experts in this domain
Formal Development Procedures	No
Type of Knowledge Representation	Production rules
Inference Mechanism	Forward chaining
Requirements Analysis	No
Approval Mechanisms	Informal
Iterative Development Process	Yes
Design Changes	Yes, to accomodate customer requests
Prototypes Built	Some critical modules
Development Time: Initial Prototype	7 months
Development Time: End Product	6 months
Documentation	Specification and informal memos
Configuration Management	No
Conventional SW Interface	Use of mailboxes for C and BLISS
Tools Used: Requirements Analysis	None
Tools Used: Development	Code generator for database routines
Tools Used: Testing	Simulator
Formal Test Procedures	None
Testing Criteria	None
Test Data	Simulator
Self Modifying Code	No

4.1 Case Study Data

Table 4.1.6 Digital Equipment Corporation - XSEL XCON System

FEATURE	APPLICATION
Problem Category	Planning and control
Domain	Computer systems configuration
Current System Phase	Production mode
Type of Funding	Internal
Experienced AI Staff	Yes - trained internally
Size of Development Team	35
Level of User Involvement	Heavy
Domain Expert Role	Multiple experts - various roles
Formal Development Procedures	Yes
Type of Knowledge Representation	Rules
Inference Mechanism	Forward chaining (OPS5 interpreter)
Requirements Analysis	Yes - problem definition
Approval Mechanisms	Informal - management team concept
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	For major mods only
Development Time: Initial Prototype	5 months
Development Time: End Product	12-18 mo. for 1st installation/quarterly upgrades now
Documentation	Architectural level, comments in code
Configuration Management	Yes
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	No
Tools Used: Development	VAX11/VMS, OPS5, DBMS + some "home-grown"
Tools Used: Testing	Yes, for code & unit test phase
Formal Test Procedures	Regression testing/Problem reporting system
Testing Criteria	Qualitative only
Test Data	Customer orders- real and hypothesized
Self Modifying Code	No

Table 4.1-7 : Expert Technologies Inc. - PEGASYS

FEATURE	APPLICATION
Problem Category	Automatic pagination
Domain	Yellow Page directories
Current System Phase	Delivered
Type of Funding	Internal
Experienced AI Staff	Yes - 3
Size of Development Team	8
Level of User Involvement	High
Domain Expert Role	Knowledge acquisition
Formal Development Procedures	
Type of Knowledge Representation	Semantic network of frames
Inference Mechanism	Heuristic search mechanism
Requirements Analysis	Yes
Approval Mechanisms	Yes, at project manager/senior engineer level
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	7 man-months
Development Time: End Product	69 man-months
Documentation	Yes
Configuration Management	Yes - proprietary
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	KEE, Knowledge Craft, TI Explorers
Tools Used: Development	Yes, at first
Tools Used: Testing	None
Formal Test Procedures	Yes
Testing Criteria	Acceptance Test Plan (ATP)
Test Data	Simulation
Self Modifying Code	No

Table 4.1-8 : Frey Associates, Inc. - THEMIS Management Information System

FEATURE	APPLICATION
Problem Category	Natural language processing
Domain	Database query
Current System Phase	Completed - commercial system
Type of Funding	Internal
Experienced AI Staff	Yes, 1 out of 12
Size of Development Team	6-12
Level of User Involvement	Medium-heavy
Domain Expert Role	Not applicable
Formal Development Procedures	No
Type of Knowledge Representation	Rule-based
Inference Mechanism	Forward chaining
Requirements Analysis	Yes
Approval Mechanisms	Informal
Iterative Development Process	Yes
Design Changes	Throughout system development
Prototypes Built	Yes
Development Time: Initial Prototype	6 months (1 man year)
Development Time: End Product	10 man years for final product
Documentation	Yes - extensive (user's manuals etc.)
Configuration Management	Yes, source code and version control
Conventional SW Interface	Yes (Fortran and user interface)
Tools Used: Requirements Analysis	None
Tools Used: Development	InterLISP
Tools Used: Testing	None
Formal Test Procedures	Yes, built-in test, regression, auto. error logging
Testing Criteria	Internal testers and user agreement with conclusions
Test Data	User defined or hypothetical queries (correct and incorrect)
Self Modifying Code	Yes

Table 4.1-9 : GTE Data Services - Central Office Maintenance Printout Analysis and Suggest System (COMPASS)

FEATURE	APPLICATION
Problem Category	Fault Diagnosis
Domain	Telecommunication switch hardware
Current System Phase	Limited field study
Type of Funding	Internal
Experienced AI Staff	2
Size of Development Team	2 - 7
Level of User Involvement	Small - Domain Expert was supervisor of end-users
Domain Expert Role	1 week per month met with knowledge engineers
Formal Development Procedures	Yes
Type of Knowledge Representation	Frames and production rules
Inference Mechanism	KEE supplied
Requirements Analysis	No
Approval Mechanisms	Informal - supervisory level
Iterative Development Process	Only during early phases
Design Changes	Some
Prototypes Built	Two
Development Time: Initial Prototype	100 man-months
Development Time: End Product	106 man-months
Documentation	Technical notes and Knowledge Acquisition Rules documents
Configuration Management	Implemented a software control knowledge base
Conventional SW Interface	None
Tools Used: Requirements Analysis	None
Tools Used: Development	KEE
Tools Used: Testing	LISP procedure to do testing in batch mode
Formal Test Procedures	No
Testing Criteria	Independent experts evaluation of COMPASS rules and output
Test Data	Hardware error message files
Self Modifying Code	No

Table 4.1-10 : IBM Federal Systems Group - **Fault Diagnosis and Resolution System (FDRS)**

FEATURE	APPLICATION
Problem Category	Fault diagnosis
Domain	Satellite ground-based maintenance
Current System Phase	Installation & test
Type of Funding	Independent research and development
Experienced AI Staff	Yes, 1
Size of Development Team	4
Level of User Involvement	None
Domain Expert Role	None
Formal Development Procedures	Yes
Type of Knowledge Representation	Production rules
Inference Mechanism	Forward and backward chaining
Requirements Analysis	No
Approval Mechanisms	Standard research approval process
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	6 Man-months
Development Time: End Product	20 Man-months
Documentation	Yes
Configuration Management	Informal
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	None
Tools Used: Development	Commercial & in-house shell
Tools Used: Testing	None
Formal Test Procedures	None
Testing Criteria	Dynamic evaluation of rule bases
Test Data	Simulation
Self Modifying Code	No

Table 4.1-11 : Inference Corp. - Authorizer's Assistant

FEATURE	APPLICATION
Problem Category	Intelligent assistant
Domain	Charge authorizations
Current System Phase	Delivered system
Type of Funding	Contract
Experienced AI Staff	Yes
Size of Development Team	8
Level of User Involvement	High
Domain Expert Role	Included in entire development process
Formal Development Procedures	Yes
Type of Knowledge Representation	Production rules
Inference Mechanism	Forward chaining
Requirements Analysis	Yes
Approval Mechanisms	Yes
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	15 Man-months
Development Time: End Product	87 Man-months
Documentation	Yes
Configuration Management	None
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	None
Tools Used: Development	ART and Symbolics development utilities
Tools Used: Testing	In-house batch test bed facility
Formal Test Procedures	No
Testing Criteria	Results compared with expert conclusions
Test Data	Sets of test cases
Self Modifying Code	No

Table 4.1-12 : Inference Corp. - Medical Charge Evaluation Control (Medchec)

FEATURE	APPLICATION
Problem Category	Fraud Detection
Domain	Medical Insurance Claims
Current System Phase	Development
Type of Funding	Contract
Experienced AI Staff	Yes
Size of Development Team	3
Level of User Involvement	High, they were the experts
Domain Expert Role	Defined requirements of system
Formal Development Procedures	Yes
Type of Knowledge Representation	Frames and production rules
Inference Mechanism	Forward & backward chaining
Requirements Analysis	Yes
Approval Mechanisms	None
Iterative Development Process	Yes
Design Changes	Yes, mostly low level ones
Prototypes Built	Yes
Development Time: Initial Prototype	6 man-months
Development Time: End Product	12 man-months (7 calendar months)
Documentation	Some
Configuration Management	None
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	None
Tools Used: Development	ART, Symbolics
Tools Used: Testing	None
Formal Test Procedures	None
Testing Criteria	No formal process
Test Data	Live data
Self Modifying Code	No

Table 4.1-13 : Lockheed Aircraft Service Company - Expert Software Pricer (ESP)

FEATURE	APPLICATION
Problem Category	Software costing
Domain	Sizing of software
Current System Phase	Complete
Type of Funding	Internal
Experienced AI Staff	No
Size of Development Team	2
Level of User Involvement	Moderate
Domain Expert Role	No expert used
Formal Development Procedures	Yes, internal
Type of Knowledge Representation	Frame based
Inference Mechanism	Backward chaining provided by LES
Requirements Analysis	Yes
Approval Mechanisms	Yes
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	4 man-months
Development Time: End Product	12 man-months
Documentation	Software Requirements Specification
Configuration Management	Yes
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	None
Tools Used: Development	LES and conventional compilers
Tools Used: Testing	None
Formal Test Procedures	None
Testing Criteria	+/- 80% of actual size and costs
Test Data	Existing systems with known size and costs
Self Modifying Code	No

Table 4.1-14 : Lockheed Aircraft Service Company - Frequency Hopper Signal Identifier

FEATURE	APPLICATION
Problem Category	Detection & characterization of frequency hopped signals
Domain	Signal identification
Current System Phase	Completed prototype
Type of Funding	Internal
Experienced AI Staff	Yes
Size of Development Team	1
Level of User Involvement	High
Domain Expert Role	Small - knowledge acquisition
Formal Development Procedures	No
Type of Knowledge Representation	Temporal framework
Inference Mechanism	Temporal logic and pattern matching
Requirements Analysis	Yes
Approval Mechanisms	No
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	9 months
Development Time: End Product	System not fielded
Documentation	Yes
Configuration Management	No
Conventional SW Interface	No
Tools Used: Requirements Analysis	None
Tools Used: Development	LISP
Tools Used: Testing	None
Formal Test Procedures	No
Testing Criteria	Consistency and improved performance
Test Data	Simulation
Self Modifying Code	No

Table 4.1-15 : Lockheed-Georgia Company - Pilot's Associate

FEATURE	APPLICATION
Problem Category	Intelligent assistant
Domain	Combat avionics
Current System Phase	Analysis
Type of Funding	Contract (US Air Force)
Experienced AI Staff	Yes 65% of the development team
Size of Development Team	Over 40
Level of User Involvement	Heavy
Domain Expert Role	Several experts involved in knowledge acquisition
Formal Development Procedures	Yes
Type of Knowledge Representation	Varied
Inference Mechanism	Varied
Requirements Analysis	Yes
Approval Mechanisms	Informal and some formal reviews scheduled
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	2 years (estimate)
Development Time: End Product	Not completed
Documentation	Yes
Configuration Management	Yes - CMS
Conventional SW Interface	Undetermined
Tools Used: Requirements Analysis	None
Tools Used: Development	ART, LES, OPS5
Tools Used: Testing	None so far
Formal Test Procedures	Yes
Testing Criteria	Satisfaction of system requirements plus Experts evaluation of system performance
Test Data	Simulation
Self Modifying Code	No

Table 4.1 16 : MITRE Inc. (Bedford) - Liquid Oxygen Expert System

FEATURE	APPLICATION
Problem Category	Fault detection/diagnosis
Domain	Space technology
Current System Phase	Final prototype complete
Type of funding	Contract (NASA)
Experienced AI Staff	Yes
Size of Development Team	2
Level of User Involvement	Heavy
Domain Expert Role	Active in system development
Formal Development Procedures	No
Type of Knowledge Representation	Frames
Inference Mechanism	Frame-based
Requirements Analysis	Yes
Approval Mechanisms	Informal - domain expert
Iterative Development Process	Yes
Design Changes	Major throughout system development
Prototypes Built	Yes
Development Time: Initial Prototype	6 months
Development Time: End Product	2 years
Documentation	Nothing formal
Configuration Management	No, did tape backups
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	No
Tools Used: Development	Yes, Symbolics Zetalisp environment
Tools Used: Testing	No
Formal Test Procedures	No
Testing Criteria	User agreement with conclusions
Test Data	Live sensor data
Self Modifying Code	No

Table 4.1–17 : MITRE Inc. (McLean) - ANALYST

FEATURE	APPLICATION
Problem Category	Interpretation/assessment
Domain	Battle management
Current System Phase	Completed - production model
Type of funding	Sponsored research with follow on contract
Experienced AI Staff	3-4
Size of Development Team	5
Level of User Involvement	Domain expert, review and test
Domain Expert Role	Active in development and transition to field
Formal Development Procedures	No, used rapid prototyping
Type of Knowledge Representation	Frames and rules
Inference Mechanism	Frame-based and goal directed
Requirements Analysis	Matching of problems to AI features
Approval Mechanisms	Informal; domain expert and program manager
Iterative Development Process	Yes, adding knowledge
Design Changes	At least one major change
Prototypes Built	One
Development Time: Initial Prototype	8 months
Development Time: End Product	18 months
Documentation	Listings, nothing formal
Configuration Management	Prior to release: build level
Conventional SW Interface	No
Tools Used: Requirements Analysis	None
Tools Used: Development	Micro-compiler for Lisp Machine/ built others
Tools Used: Testing	No
Formal Test Procedures	No
Testing Criteria	Domain expert and user satisfaction
Test Data	Simulated data stream
Self Modifying Code	No

Table 4.1 18 : Northrop Aircraft Division - Expert System for Target Attack Sequencing (ESTAS)

FEATURE	APPLICATION
Problem Category	Decision aids
Domain	Combat avionics
Current System Phase	Completed feasibility prototype
Type of Funding	Internal
Experienced AI Staff	Yes
Size of Development Team	4
Level of User Involvement	Active, throughout development and testing
Domain Expert Role	Very active for knowledge acquisition and testing
Formal Development Procedures	Yes, conventional framework
Type of Knowledge Representation	Production rules
Inference Mechanism	Forward & backward chaining
Requirements Analysis	Yes
Approval Mechanisms	Yes, frequent reviews and several demos
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	6 months
Development Time: End Product	Not applicable
Documentation	Yes, enforced
Configuration Management	Yes
Conventional SW Interface	Not at this time
Tools Used: Requirements Analysis	None
Tools Used: Development	LISP workstations using Common Lisp
Tools Used: Testing	Yes, built-in trace facilities
Formal Test Procedures	No
Testing Criteria	User satisfaction/expert assessment
Test Data	Usage
Self Modifying Code	No

Table 4.1-19 : PAR Government Systems Corporation - Cost Benefit of Tactical Air Operations (CBTAO)

FEATURE	APPLICATION
Problem Category	Decision Aid
Domain	Cost/benefit of tactical missions
Current System Phase	Completed prototype
Type of Funding	Contract
Experienced AI Staff	Yes
Size of Development Team	3-6 computer scientists & in-house expert(s)
Level of User Involvement	Moderate
Domain Expert Role	Define the problem, Knowledge engineering and test prototype
Formal Development Procedures	Yes
Type of Knowledge Representation	Inference network of production rules with confidence factors
Inference Mechanism	Goal-directed
Requirements Analysis	Yes
Approval Mechanisms	Informal - project staff
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	24 man-months (including preliminary investigation phase)
Development Time: End Product	System not fielded
Documentation	Yes, extensive
Configuration Management	Yes
Conventional SW Interface	Yes, support environment
Tools Used: Requirements Analysis	None
Tools Used: Development	In-house developed Expert System Shell and graphics tools
Tools Used: Testing	No
Formal Test Procedures	Yes
Testing Criteria	Experts and potential users ratings of system
Test Data	Scenario
Self Modifying Code	No

Table 4.1 20 : PAR Government Systems Corporation - Duplex Army Radio/Radar Targeting Decision Aid (DART)

FEATURE	APPLICATION
Problem Category	Decision Aid
Domain	Target identification/classification
Current System Phase	Completed prototype
Type of Funding	Contract
Experienced AI Staff	Yes
Size of Development Team	3-6 computer scientists & in-house expert(s)
Level of User Involvement	Moderate
Domain Expert Role	Define the problem, knowledge engineering and test prototype
Formal Development Procedures	Yes
Type of Knowledge Representation	Inference network of production rules with confidence factors
Inference Mechanism	Goal-directed
Requirements Analysis	Yes
Approval Mechanisms	Informal - project staff
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	36 man-months (including preliminary investigation phase)
Development Time: End Product	System not fielded
Documentation	Yes, extensive
Configuration Management	Yes
Conventional SW Interface	Yes, user interface and support environment
Tools Used: Requirements Analysis	None
Tools Used: Development	In-house developed Expert System Shell
Tools Used: Testing	No
Formal Test Procedures	Yes
Testing Criteria	Experts and potential users ratings of system
Test Data	Test scenario
Self Modifying Code	No

Table 4.1-21 : PAR Government Systems Corporation - See and Project Enemy Activity (SPEA)

FEATURE	APPLICATION
Problem Category	Decision aid
Domain	Battle situation projections
Current System Phase	Completed prototype
Type of Funding	Contract
Experienced AI Staff	Yes
Size of Development Team	3-6 computer scientists & in-house expert(s)
Level of User Involvement	Moderate
Domain Expert Role	Define problem and test system
Formal Development Procedures	Yes
Type of Knowledge Representation	Object oriented
Inference Mechanism	Inheritance
Requirements Analysis	Yes
Approval Mechanisms	Informal - project staff
Iterative Development Process	N/A
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	30 man-months
Development Time: End Product	System not fielded
Documentation	Yes, extensive
Configuration Management	Yes
Conventional SW Interface	Yes, databases and support environment
Tools Used: Requirements Analysis	None
Tools Used: Development	Flavors
Tools Used: Testing	None
Formal Test Procedures	Yes
Testing Criteria	Experts and potential user ratings of system
Test Data	Scenario
Self Modifying Code	No

Table 4.1-22 : Sanders Associates - Test Assistant (TESS)

FEATURE	APPLICATION
Problem Category	Test equipment assistant
Domain	Countermeasure systems
Current System Phase	Mature prototype - funding cut
Type of funding	Internal Research & Development
Experienced AI Staff	None
Size of Development Team	3
Level of User Involvement	Low
Domain Expert Role	Knowledge definition
Formal Development Procedures	No
Type of Knowledge Representation	Frames
Inference Mechanism	Frame-based and constraints
Requirements Analysis	No
Approval Mechanisms	Informal
Iterative Development Process	Yes
Design Changes	Yes, user interface and knowledge base
Prototypes Built	Yes
Development Time: Initial Prototype	6 months
Development Time: End product	Not applicable
Documentation	IR&D plan, listings, frame knowledge
Configuration Management	No
Conventional SW Interface	No
Tools Used: Requirements analysis	No
Tools Used: Development	Symbolics environment & some internally developed
Tools Used: Testing	No
Formal Test Procedures	No
Testing Criteria	User satisfaction
Test Data	Daily usage
Self Modifying Code	No

Table 4.1–23 : Schlumberger, Inc. - Dipmeter Advisor System

FEATURE	APPLICATION
Problem Category	Interpretation
Domain	Geology - dipmeter logs
Current System Phase	Completed - commercial system
Type of Funding	Commercial
Experienced AI Staff	Unknown
Size of Development Team	Questionable 4-6 (assumption)
Level of User Involvement	Active in system definition
Domain Expert Role	Defined Dipmeter system knowledge
Formal Development Procedures	No
Type of Knowledge Representation	Rule-based
Inference Mechanism	Forward chaining
Requirements Analysis	Yes
Approval Mechanisms	Informal (domain expert)
Iterative Development Process	Yes
Design Changes	Major throughout system development
Prototypes Built	Yes
Development Time: Initial Prototype	18-22 months
Development Time: End Product	4 years
Documentation	Informal (Assumption)
Configuration Management	Unknown
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	Yes
Tools Used: Development	Yes: Strobe, Impulse, InterLISP
Formal Test Procedures	Yes
Testing Criteria	User acceptance; integration testing with the system
Test Data	Graphical interface with scrolling log data
Self Modifying Code	Unknown

4.1 Case Study Data

Table 4.1 24 : Software Architecture and Engineering, Inc. - Decision Support System

FEATURE	APPLICATION
Problem Category	Decision support
Domain	Collaborative decision information support
Current System Phase	Development
Type of Funding	Contractual
Experienced AI Staff	1
Size of Development Team	2
Level of User Involvement	Very active
Domain Expert Role	Requirements definition. Analysis
Formal Development Procedures	Yes, internal
Type of Knowledge Representation	Rule-based
Inference Mechanism	Production system
Requirements Analysis	Yes
Approval Mechanisms	Mostly informal except requirements. CM approval for changes
Iterative Development Process	Yes
Design Changes	Yes
Prototypes Built	Yes
Development Time: Initial Prototype	12 months
Development Time: End Product	Not applicable
Documentation	Yes, requirements document, design document, increment plan
Configuration Management	Yes
Conventional SW Interface	Yes
Tools Used: Requirements Analysis	Text processing programs
Tools Used: Development	DSDS (Decision Support Development System)
Tools Used: Testing	None
Formal Test Procedures	Alpha and beta testing
Testing Criteria	Experts made final decision on system performance
Test Data	Usage
Self Modifying Code	No

Table 4.1-25 : Software Architecture and Engineering, Inc. - Sensitive Financial Analysis System

FEATURE	APPLICATION
Problem Category	Classification
Domain	Financial analysis
Current System Phase	Completed - production system
Type of Funding	Contractual
Experienced AI Staff	1
Size of Development Team	2
Level of User Involvement	Active
Domain Expert Role	Involved in list process, defined schedule
Formal Development Procedures	Yes, internal
Type of Knowledge Representation	Rule-based
Inference Mechanism	Production system
Requirements Analysis	Yes, internal review
Approval Mechanisms	No
Iterative Development Process	Yes, incremental system development
Design Changes	No
Prototypes Built	Yes
Development Time: Initial Prototype	3 months
Development Time: End Product	8 months
Documentation	Not required by customer. Internal notes.
Configuration Management	Informal
Conventional SW Interface	Interface with conventional software via Apollo OS commands
Tools Used: Requirements Analysis	None
Tools Used: Development	KES, text editor and KES parser
Tools Used: Testing	None
Formal Test Procedures	Case studies with known outcomes
Testing Criteria	Performance judged by experts
Test Data	Test case studies, individual rules
Self Modifying Code	No

4.1 Case Study Data

Table 4.1-26 : Texas Instruments Inc. - Production Scheduler Project

FEATURE	APPLICATION
Problem Category	Scheduling mechanism
Domain	Textile fiber production and inventory
Current System Phase	Project terminated prior to viable prototype
Type of Funding	Commercial
Experienced AI Staff	Yes
Size of Development Team	3
Level of User Involvement	Minimal
Domain Expert Role	Knowledge and constraint definition.
Formal Development Procedures	No
Type of Knowledge Representation	Frame-based
Inference Mechanism	Algorithmic mechanism controlled schedule processing
Requirements Analysis	No
Approval Mechanisms	None
Iterative Development Process	Not applicable
Design Changes	Yes
Prototypes Built	No
Development Time: Initial Prototype	Not applicable
Development Time: End Product	Not applicable
Documentation	Informal memos
Configuration Management	Stamp of saved Lisp source code files
Conventional SW Interface	No
Tools Used: Requirements Analysis	None
Tools Used: Development	Lisp, windowing forms management tool
Tools Used: Testing	None
Formal Test Procedures	Planned but not applicable
Testing Criteria	Interface, schedule checked for accuracy. Both tested together
Test Data	Schedule, user interface
Self Modifying Code	No

4.2 Evaluation of the Case Study Data

This subsection presents an analysis of the case study data depicting generally common aspects, relational trends and distinguishing features.

4.2.1 Common Aspects

The twenty-six case study tables were evaluated as a single group in an attempt to identify commonalities germane to the development process regardless of features such as current project phase, problem category or domain. In general, the projects appear to share a number of common characteristics as delineated below:

- Successful systems;
- AI experience;
- Small development teams;
- Participative users;
- Knowledge representation schemes;
- Iterative development process;
- Rapid prototyping;
- Use of development tools; and
- Static code.

All participants claimed that their systems were successful as measured by some criteria such as user satisfaction, expert evaluation, productivity gains and/or demonstrating the feasibility of a new technology. In addition, the project teams were generally comprised of 5 or less people and included at least one engineer with AI experience. The users generally played an active role in system development and their acceptance was extremely important in determining whether or not the system was a success.

The development process for all projects was iterative, including design changes and prototyping. For many of the projects, the initial prototype was completed in 6 months or less. The knowledge representation scheme for all of the projects was based on rules, frames or a combination of both.

Most of the twenty projects used tools during the development phase. Some tools were built by the participants but many of the tools used were inherent to the AI workstation. Tools were generally not used in support of requirements analysis or testing.

Lastly, evaluation of the case studies indicates that few systems include self-modifying code

4.2.2 Relational Trends

The case study data was also analyzed in separate groupings as a function of certain relational characteristics. Those groupings that appeared to indicate significant trends are:

- team size versus tools;
- IR & D versus commercially funded projects; and
- user involvement as a function of project phase.

These relationships and apparent features are discussed in the following subsections.

4.2.2.1 Team Size Versus Tools

The literature review indicated that both tool power and development team size were areas of difference between AI and conventional software development. Analysis of responses to the questionnaire and further review of reports from AI development teams indicates there is a relationship between the two characteristics. As noted earlier, the trend has been to provide tools of increased power to accomplish specific functions after knowledge was acquired about the task. Tools have also been developed to shorten system development time, to allow small teams of people to solve still larger problems, and to automate the development process ensuring some higher degree of confidence in the consistency, quality, or reliability of the programs.

Some very specialized tools were developed or purchased to perform specific system functions such as data base interfaces, or micro-code compilers. Most tools were developed to support the development team efforts and were aimed at a better program implementation. There were some tools used to assist in the testing effort. There were still fewer tools used in the fielded or mature production phases of the KBS systems.

In general, the tools supported the development team processes. Tool usage was not as prevalent in the other stages of a project. Fewer tools were purchased than were developed. More tools were developed for IR & D projects than for contract projects. Tools developed in the academic community were adapted for use in IR & D projects.

4.2.2.2 IR & D Versus Contract Funding

Of the twenty-six case studies, twelve were funded via internal research & development money, nine were commercially contracted and five contracted by the DOD. This cross-section of the data revealed some interesting trends within the three groups.

Among the IR & D projects, several characteristics were observed. Namely, these projects more often followed structured development procedures and produced more written documentation than the commercially funded projects. In addition, initial prototypes were generally completed earlier (within 6 months) when compared to the commercial systems. These traits seem to be commensurate with obtaining and prolonging management support. The onus is on the project staff to earn managerial support by showing enough progress to satisfy them at specified time intervals.

Among the commercial projects, expert involvement seemed to be greater when compared to the IR & D projects. In addition, most all of the commercial systems were developed to some end state whereas many of the IR & D projects were terminated prior to completion. It is not surprising that in the commercial sector, there is a greater commitment towards developing an end product and not as much competition for the same pool of funds once the contract has been awarded.

The projects contracted by the DOD were similiar to the commercial systems. Along with displaying the above mentioned characteristics, they also produced the most written documents of the three groups. They tended to use some form of standard, such as the phased waterfall development approach or a formal MIL-STD specification to design the system. If not adhered to strictly, formal standards were at least used as a guideline for system documentation.

4.2.2.3 User Involvement Versus Project Phase

Observations made from the case studies indicate that user and domain expert involvement are both critical to the success of KBS software development. The user's evaluation of a system determines whether continual financial commitment can be obtained throughout the development effort and helps the user develop realistic expectations of ultimate system functionality and performance. Users tend to actively participate throughout the entire system development effort. However, according to the study, the user's involvement appears to increase from the feasibility to the development phase and continue into production particularly with respect to testing the system.

The domain expert's knowledge seems to be mainly required during feasibility and development of the prototype in order to define the requirements, the system's knowledge, and the problem-solving techniques. The expert also helps with the testing effort.

4.2.3 Distinguishing Features

In contrast to common aspects and relational trends, several of the cases reveal features that are not widely reported among the study participants. One distinct characteristic has to do with the use of audio/visual equipment during the development process. Specifically, during the knowledge acquisition phase, both Northrop Avionics Division and Schlumberger-Doll Research tape recorded interview sessions with the experts. The information reported was then captured for repetitive referencing and less subject to recollection augmented by the knowledge engineer's notes alone. In addition, the inflections used by the experts are maintained which may often affect the knowledge engineer's interpretation of the data.

In assessing user satisfaction with DSDS, SA & E video taped customer reactions while using the product. This approach was reported to be more conducive than asking the users questions about performance while they are simultaneously acquainting themselves with a new system.

Lockheed-Georgia Company's Pilot's Associate is being developed by a much larger team than the other projects in the survey. The distinguishing characteristic is not necessarily the size of the team though. A more prominent feature is that several expert systems are integrated into the product. Therefore the number of personnel working with each individual expert system would most likely be similar to the size of the development teams of the other projects.

The development team for DEC's XCON system is also much larger than the others. The notably large size of the knowledge base (more than 10,000 rules) requires much support. Several knowledge engineering teams, each comprised of seven to eight people, perform knowledge acquisition and representation tasks, continually increasing the size of the knowledge base as new products are developed and existing products are modified. Groups to manage administrative duties, user support, and technical support also exist to provide assistance.

In terms of managing a large systems project, Northrop Avionics Division strongly recommends a conventional software development cycle wherein iterations are acceptable. This technique would involve the required engineering disciplines and implies that the engineers across the various disciplines must be knowledgeable in AI. Because a military systems outlook is so important, knowledge engineers who are not experienced with large avionics systems development are inadequate for the job unless the problem is well-defined and well-bounded.

4.3 SDI Related Issues/Implications

The twenty-six case studies provide an experience base from which SDI related issues and implications can be drawn. The most notable issues are acquisition risk reduction, tools, real-time processing requirements, project management and testing.

In terms of reducing the risks associated with acquiring a KBS system, it is prudent to examine those aspects that are common to all case studies. Namely, the development process for all twenty-six projects was iterative and based on prototyping. With this method, the requirements specification and the design are expected to change over some period of time with the ultimate goal of providing a system that satisfies the users needs. In addition, it was noted that all of the case projects employed a common set of knowledge representation schemes: rules, frames (or frame-like) or a combination of both. The knowledge representation techniques for rules and frames seem stable and would not present risk in the application of this technology.

In the case studies, reasoning strategies have centered on search and inferencing techniques. There are several commercially available inference engines and LISP embedded language capabilities (Section 3.4.5) that have been used successfully. Extensive experience with the tools in a development environment has led to a reduction in risk associated with their use for software development. Tool use in other phases of a KBS project has not been extensive and represents an area where more effort needs to be directed.

Since all the case study participants claimed success in some manner, it may be wise to scrutinize deviations from the above techniques during the KBS system acquisition process.

An important issue involves the nature of real-time processing that is likely to be required in SDI applications. There is only one system that was represented as having real-time requirements commensurate with SDI needs, and it was never tested to verify that it met those requirements. The requirements on KBS software in the SDI system are likely to be at least an order of magnitude greater than for a system where the operator interaction is the pacing element for real-time.

With two exceptions, development teams have been uniformly small and focused on a single project that was scoped to fit their capabilities. It is expected that the BM/C^3 AI software will be larger

and more complex than any of the projects reported on. There are questions about how to partition the system so that several teams can work on the problem, and ensure that the functional interfaces will work. Project management is also an issue since there is no experience available in terms of managing a very large KBS system. Application of tools to support management decisions in the development of a system also needs further study.

The testing aspect of expert system development is not well-defined because of the nondeterministic nature of the solution space. Consequently, those case systems that pertain to military applications are intelligent assistants or decision aids as opposed to autonomous systems. In addition, for all of the cases, the test data is based on either hypothesized inputs or actual usage. In terms of SDI applications, the testing aspect has several implications as delineated below.

The nature of a decision aid is that the responses or conclusions reached are not 100% accurate. Because human intervention is expected in reaching a final decision, some degree of system inaccuracy is acceptable. For SDI applications, however, the service period of the system is expected to be so short that there will be little possibility of human intervention [49, Parnas].

In addition, heuristic programs are often developed by trial and error using the concept of *build a little, test a little.*When errors are encountered, the expert is asked to review the situation and add more knowledge. Because heuristic programs may exhibit important gaps in knowledge at unexpected times, this approach would not be acceptable for many SDI applications.

Furthermore, the best expert system simply imitates an expert exceptionally well. Since humans are not perfect, expert systems cannot be expected to reach accurate conclusions all of the time.

In terms of test data, it would be impossible to test an SDI system during actual use. Furthermore, the set of cases that could be hypothesized would not come close to covering the set of all possible situations. The software systems addressing SDI objectives will be significantly larger and more complex than any systems built to date. Consequently, the behavior of these systems under all conceivable circumstances cannot be known in advance. Another major factor complicating the prediction of potential stimuli is the use of enemy countermeasures of which there is no current knowledge. A significant aspect that has not been addressed is how to test a system for robustness in the face of uncertainty.

As indicated by the commentary in this subsection, continuing research is required in terms of the application of KBS to the SDI. More details are presented in Section 5.3.

SECTION 5

Synopsis

5.1 Comparison of KBS Development Process to DOD-2167

The software development model under DOD-STD-2167 is patterned after the waterfall model previously discussed and includes the concepts of activities, products, reviews and baselines to further expand the waterfall concept. The model represents software development from the government viewpoint, and as such visualizes the development process as a series of sequential phases with reviews and documentation integral to each particular phase. It is generally recognized by both government and industry that software development is not a well-bounded sequential process but instead consists of overlapping phases.

The 2167 waterfall model was developed to provide government insight into software development progress and to provide control mechanisms over the evolving product as development occurred. There was a strong configuration management influence in developing the model which resulted in emphasis being placed on configuration identification through the concept of computer software configuration items(CSCI's), and configuration control. Although 2167 provides no guidance on *what constitutes a CSCI, a companion document, MIL-STD-483, does provide guidance concerning* CSCI selection. The 2167 requirements for baseline establishment and control provide the mechanism for controlling requirements, design, and code as it evolves. Associated with the establishment of baselines are various reviews aimed at assessing software development progress at various phase points (e.g., requirements analysis, preliminary design) and determining readiness for baseline control. The documentation produced was designed to be a natural fall-out of the activities within a particular phase of development.

This ordered, phased approach was expected to improve the DOD posture for developing and supporting software. The documented 2167 model was designed to provide: visibility and control over the evolving software products; a quality product at delivery; and a proper environment for maintenance and modification of the software in its fielded environment. Although 2167 is a new standard, previous developments that have followed the principles of 2167 have generally been successful developments and have provided software that is maintainable and supportable when fielded.

As stated previously, the KBS system development process is a highly iterative process and because of this most models represent development as an incremental process reflecting *build a little, test a little* implementation. Models developed for expert systems, for example, clearly define the AI system development process as a continuous one [35,59, Hayes-Roth,Scown] with feedback to any of the previous phases of development. Contrast this with the phased development of software under DOD-STD-2167 and it is obvious that there exists incompatabilities that are of significance.

These incompatabilities are not significant in terms of activities performed during development but instead are mainly related to the use of prototyping KBS's as discussed in Section 3. Whereas the

2167 model and its use in the development of conventional software is essentially oriented towards defining the total requirements, then producing a design followed by complete implementation, the KBS approach to system development advocated by most developers follows a different planned approach. KBS systems typically define the problem domain, then begin by designing, building and testing some small subset of this domain. Feedback from this first version of the software can then be incorporated into the next increment with its added capabilities This process continues until the complete domain has been implemented or in many cases never ends as further domain knowledge is collected.

This incremental development approach, when viewed from a documentation, review/audit, and baseline management perspective, leads to the need for a new look at the management and technical control mechanisms associated with KBS system development. Questions to be considered are: At what point in the development process are controls established; What reviews are conducted and when; and What documentation is produced and when? The incremental nature of the KBS development process must be factored into providing answers to these questions. Further, it is expected that future SDI system components will include both conventional and KBS software. The presence of both types of software will require that the development approaches, attendant reviews/audits, configuration controls, and documentation be compatable. The model for the KBS development process is a part of the Volume II report.

5.2 Interface of Conventional and KBS Software

Interface of KBS with conventional software is focused on two areas of major interest: conventional programs written in Ada and data bases. The language Ada is chosen to represent conventional software because of its standardization and projected usage rate in large systems of interest. Data bases are chosen as another focal point because of the obviously important role they would play in any battle management system of significant size. The following sections discuss the management, functional, testing and integration aspects of the interface issues between KBS and conventional software.

5.2.1 Management Perspectives

There are critical issues associated with the sequencing development of both KBS and conventional software. Management planning, reviews and direction could take different forms depending on whether the KBS software will be retrofitted, integrated during development or planned for forward fit with a conventional software system.

Bridging the differences between the different and possibly competitive technologies, methodologies and capabilities requires communications and contract mechanisms that are appropriate to the task. Implicit in the communications and contracts among the cooperating groups is the creation of some sort of meaningful intermediate products that can be used to measure performance, risk, and integration

Cost management will be complicated due to the risk factors associated with both the development of KBS software and its integration with conventional software. Costing algorithms that have been

applied to conventional software development with some success have not been applicable to the KBS development efforts. The integration of both software types in a single system may also subtly change the complexity level so the costing algorithms are no longer valid approximations to even the conventional efforts cost.

The integration of KBS and conventional software in a system will require the extension or tailoring of familiar management procedures to accomodate the characteristics of differing development styles. One of the key items in the extended procedures will be the identification of products that can be used to track the expected project cost, schedule, performance, and accomplishment goals versus the actual progress.

5.2.2 Implementation Perspectives

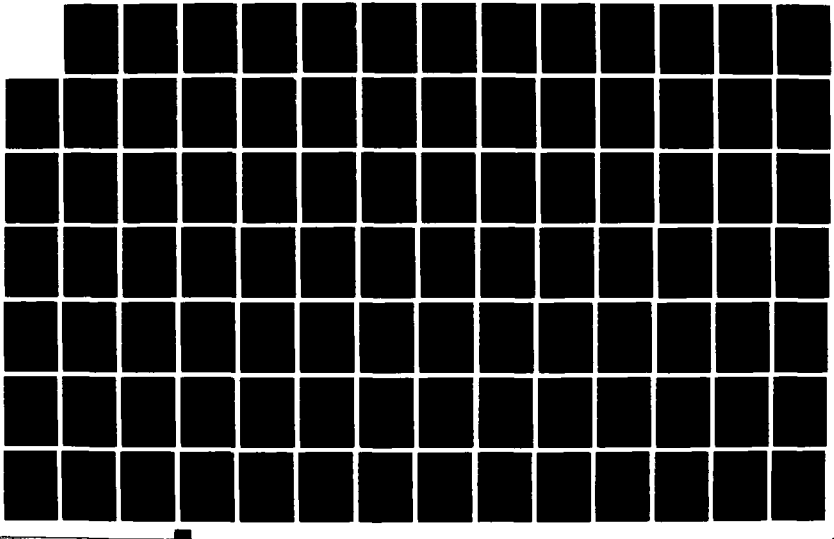
Issues in the actual implementation of KBS and conventional software include specification, design, interfaces and integration. There are additional decisions to be made in the systems engineering allocation process. With the introduction of KBS, there has to be an assignment of intelligence and knowledge that must be shared by the user of the system and the computer system. System level controls, feedback and interface will have to undergo more cycles of iteration than is common for conventional software until there are adequate engineering guidelines and knowledge to breakout these functions in a cookbook manner.

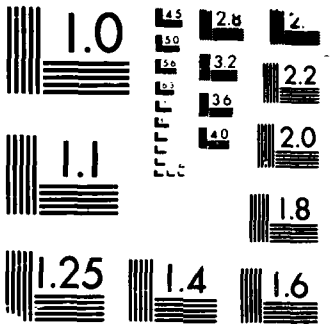
There are top level design issues including synchronization and execution of programs and handling of the data and information flows. Using present technology, the questions about control flow apply predominantly to Ada program interfaces. With the introduction of smart memories or data base machines into the system architecture, control flow would become a prominent question for data base interfaces as well. Specific questions on program controls are phrased as follows. How do the Ada and KBS processes know when to start, stop, and synch or at least align in a common reference frame? The questions about data and information flow apply to the Ada and data base interfaces equally and include: What mechanism is required to handle indeterminant sized amounts of information or data? What do the data interfaces look like and how can they be defined?

There are detailed technical questions centering around language capabilities and limitations. These questions concern computer language translation and expressivity, integration, task allocation and efficiency. Examples of issues include: Ada execution of translated KBS programs, the accommodation of self-modifying programs; control passing to possibly non-terminating programs and [illegible] tasking for concurrent solution searches.

Integration of large dissimilar programs historically has been troublesome. The [illegible] software fault tolerance capability, and the most robust of comunications and [illegible] will be required to prevent more serious difficulties. Stability of interfaces between [illegible] software is one means of ensuring better communication both within the technical [illegible] between programs. This interface definition might be one of the first items [illegible] under management control.

AD-A194 104 ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM 2/3
VOLUME 1(U) SANDERS ASSOCIATES INC NASHUA NH
C BARDAWIL ET AL. DEC 87 RADC-TR-87-249-VOL-1
UNCLASSIFIED F30602-85-C-0254 F/G 12/5 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963 A

5.2.3 Testing and QA Perspectives

The integration of KBS functionality with conventional software raises many additional questions. These can be grouped into the following broad classes. How should nondeterministic processes be tested? In expert systems, how do experts become certified? With ill defined requirements how can testing procedures be derived?

The nature of the application may itself add a dimension of complexity to testing and QA if the proposed operational environment has many unknowns that can affect the nature of the system or its responses. Quality Assurance of a system that is self modifying will require new standards and techniques.

5.2.4 Comparison of Development Techniques

Contrasts and similarities are noted between the development approach, and the intermediate and final products from KBS and Ada developed programs that have significance for their system level integration.

Ada developed programs rely on support from the language, standards for development, management and review, along with modern software engineering practices. KBS programs rely on support, tools, and power from the languages used, highly trained/skilled developers with high standards of integrity and development paradigms that have no analog in conventional practices. There are different problems inherent in the milieu of KBS than those in conventional software environments. These include dynamic growth of the data structure, control of the operating system, and indeterminancy of results.

Each development group would feel more comfortable using its native language for program development unless the application was more naturally expressed in the other language. There have only been preliminary studies in the area of productivity and release error rate. The most recent available study used a controlled experimental method to evaluate relative productivity on the four programming tasks of pattern matching, maze solution, frame editing, and a heap sort [32, Hattori]. The experimental results indicate doubled productivity for LISP over Ada. The error rate reported in this study was slightly lower for LISP. To minimize language related problems, management needs to enforce the necessary standard coding conventions and a rigorous walkthrough and review process to eliminate undocumented tricks or bizarre coding.

There are several similarities which are recognizable to practitioners in the different groups. KBS techniques clearly build on abstractions. The languages used in KBS efforts are more powerful, so smaller units of code are required for most classes of functions. When a team of people work at a single terminal, there is an aspect of code walkthrough. Implementation of the *chief programmer team concept* is necessary to avoid team members stumbling too far afield. The KBS approach to development is recognizably *build a little, test a little.*

Use of a contract basis for data representation, protocol, budgeting of computer system resources and retention of program execution control within the conventional software package are methods that could be used by management to ensure that the programs can be successfully integrated.

Just as the system engineering process now allocates the system functionality to hardware and software, there will have to be a similar process in the allocation of function and tasking to modules

with or without intelligence as well as some specification of the level of intelligence that is required. Recognition of this aspect would interface the KBS and conventional software development teams with the system engineering team to produce a working level requirements and specification document.

5.2.5 Integration with Data Bases

Integration of KBS software and data bases is critical because of the large amounts of real world data that are accessed by BM/C^3 systems of the size contemplated for SDI. The two critical issues to effective integration of the processes are: efficiency in data flow from the data base to the KBS application; and the effective direction and control of the data base by the KBS application. It is expected that the strategies to effect these implementations would be different depending on the data base model, the domain of application and the particular type of KBS software.

5.2.6 Management Implications

From a management viewpoint, not enough is known about the KBS development process to have any rules of thumb for gauging how well development is proceeding. The identification of meaningful milestones in the KBS development stream are difficult to assess due to the relative inexperience in management control of developments in this technology. This same lack of experience applies to issues across the board such as the contract agreement process, standardization, and manpower planning.

The focus of the statement of work in this area is in the identification of a standard for KBS software interfacing with conventional software, and in the management implications of intragroup activities. These software interfaces include the management and distribution of data, control and intelligence.

5.3 Application of AI to SDI Issues

The application of AI technology to SDI BM/C^3 appears to be concentrated into two areas: applications and support tools. SDI studies conducted to date have primarily concentrated on the support tools aspect with very little definition in the applications area.

The application of AI to support tools principally affects the areas of planning systems and knowledge based software assistance (KBSA). The KBSA concept of providing an automated assistant to support software developers is envisioned as a tool that will contribute significantly to software productivity improvements.

In the applications area, the potential for applying AI is only limited by the risk envisioned in developing a particular application. SDI studies to date have left the specific application of AI as a *to be defined* item. However, extensive research and development work is underway in the areas of decision support aids, expert maintenance systems, knowledge based signal processing and robotics. All of these research efforts have the potential of providing significant contributions to SDI BM/C^3.

Bibliography

[1] Addison, E.R. "Design Issues for a Knowledge-Based Controller for a Track-While-Scan Radar System". April 1986.

[2] Apte, C. and Weiss, S. *"A Knowledge Representation Framework for Expert Control of Interactive Software Systems"*. Technical Report, Rutgers University Department of Computer Science, 1984.

[3] Bachant, Judith and McDermott, John. "R1 Revisited: Four Years in the Trenches". *The AI Magazine*, Vol. 5, No. 3, pp. 21–32, Fall 1984.

[4] Balaban, David J. and Nelson, David O. "Flat is Not Necessarily Good". April 17, 1985. Lawrence Livermore National Laboratory.

[5] Barstow, David R. "Artificial Intelligence at Schlumberger". *The AI Magazine*, Vol. 6, No. 4, pp. 80–83, Winter 1985.

[6] Boehm, B.W. "Software Engineering". *IEEE Transactions on Computers*, Vol. C-25, No. 12, pp. 1226–1241, December 1976.

[7] Bonasso, R.P. Jr. "Analyst, MTP-83W00002, Contract F19628-84-C-0001". February 1984. Internal report, MITRE Corp.

[8] Booch, Grady. *"Software Engineering with Ada"*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1983.

[9] Boose, J. H. "A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology". *International Journal of Man Machine*, Vol. 23, No. 5, pp. 495 626, November 1985.

[10] Brachman, Ronald J. and Levesque, Hector J. "Frame Representations and the Declarative/Procedural Controversy". 1985.

[11] Bradley, S. and Buys, R. and ElSawy, A. and Sipes, A. "Developing a Microcomputer based Intelligent Project Planning System". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[12] Buchanan, Bruce G. and Shortliffe, Edward H. *"Rule-Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project."*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1985.

[13] Chandrasekaran, B. "Generic tasks in Knowledge based reasoning: Expert Systems at the right level of abstraction". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[14] Cohen, Paul R. and Grinberg, Milton R. "A Theory of Heuristic Reasoning about Uncertainty". *The AI Magazine*, Vol. 4, No. 2, pp. 17-24, Summer 1983.

[15] Davis, Randall. *"Expert Systems: Where Are We? and Where Do We Go From Here?"*. Technical Report, MIT Artificial Intelligence Laboratory, 1982.

Bibliography

[16] Drastal, G. and DuBois, T. and McAndrews, L. and Straguzzi, N. and Raatz, S. "Economy in Expert System Development: Aegis Combat System Maintenance Advisor". April 1986.

[17] Druffel, L.E. and Kernan, Joseph E. and Paige, K.K. and Riski, William A. "Report on the DoD Task Force on Software Problems". 1982. Third Draft, July 15.

[18] Engleman, C. and Berg, Charles and Bischoff, Miriam. "KNOBS: An Experimental Knowledge Based Tactical Air Mission Planning System and a Rule Based Aircraft Identification Simulation Facility,". In *In Proceedings of the International Joint Conference on Artificial Intelligence* , 1979.

[19] Fikes, R. and Kehler, T. "The Role of Frame-Based Representation in Reasoning". *ACM*, Vol. 28, No. 9, pp. 904-920, September 1985.

[20] Freiling, Mike and Alexander, Jim and Messick, Steve and Rehfuss, Steve and Shulman, Sherri. "Starting a Knowledge Engineering Project: A Step-by-Step Approach". *The AI Magazine*, Vol. 6, No. 3, pp. 150-164, Fall 1985.

[21] Gallant, John. "ADL Putting AI Technology to Work". *Computerworld*, Vol. 19, No. 13, pp. 45-50, April 1985.

[22] Gates, K.H. and Adelman, L. and Lemmer, J.F. "Management of AI System Software Development for Military Decision Aids". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[23] Genesereth, M. and Ginsberg, M. "Logic Programming". *ACM*, Vol. 28, No. 9, pp. 933-941, September 1985.

[24] Gilmore, J. F. and Pulaski, K. "Comparative Analysis of Expert System Tools". In *Applications of Artificial Intelligence II*, April 1986.

[25] Goldberg, R.N. and Weiss, S. M. *"An Experimental Transformation of a Large Expert Knowledge"*. Technical Report, Rutgers University, Department of Computer Science, 1980.

[26] Goyal, Shri and Prerau, David and Lemmon, Alan and Gunderson, Alan and Reinke, Robert. "COMPASS: An Expert System for Telephone Switching Maintenance". In *Expert Systems in Government Symposium*, October 24-25 1985.

[27] Green, P.E. "Resource Limitation Issues in Real-Time Intelligent Systems". April 1986.

[28] Hankins, G. B. and Jordan, J.W. and Katz, J.L. and Mulvihill, A.M. and Dumoulin, J. N. and Ragusa, J. "Expert Mission Planning and Replanning Scheduling System". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[29] Harmon, Paul and King, David. *"Expert Systems, Artifical Intelligence in Business"*. John Wiley & Sons, New York, NY, 1985.

[30] Hart, Anna. "Knowledge Elicitation: Issues and Methods". *Computer Aided Design*, Vol. 17, No. 9, pp. 455-462, November 1985.

[31] Hart, Peter. "Talks about Expert Systems". *IEEE Expert*, Vol. 1, No. 1, pp. 96-99, Spring 1986.

[32] Hattori, F. and Kushima, K. and Wasano, T. "A Comparison of LISP, PROLOG,and Ada Programming Productivity in AI Area". In *Eighth International Conference in Software Engineering*, Aug 28–30 1985.

[33] Haugeland, John. *"Artificial Intelligence: The Very Idea"*. MIT Press, Cambridge, Massachusetts, 1985.

[34] Hayes-Roth, F. "Rule–Based Systems". *ACM*, Vol. 28, No. 9, pp. 921–932, September 1985.

[35] Hayes-Roth, Frederick, and Waterman, Donald A. and Lenat, Douglas B. *"Building Expert Systems"*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1983.

[36] Kaplan, Jerrold. "The Industrialization of Artificial Intelligence: from By-line to Bottom Line". *The AI Magazine*, Vol. 5, No. 2, pp. 51-57, Summer 1984.

[37] Keller, R. *"A Survey of Research in Strategy Acquisition"*. Technical Report, Rutgers University, Department of Computer Science, April - July 1982.

[38] Kline, Paul and Dollins, Steven. *"Choosing Architecture for Expert Systems"*. Technical Report, RADC Technical Report TR-85-192 October, 1985.

[39] Levine, A.P. "ESP: Expert System for Computer Performance Management". April 1986.

[40] McGraw, Karen L. and Bruce A. "The Phantom Crew AI in the Cockpit". *DSE/E Defense Science & Electronics*, Vol. 4, No. 11, pp. 44-54, November 1985.

[41] McLaren, R.W. and Lin, H.Y. "Knowledge-Based Approach to Ship Identification". April 1986.

[42] Metzger, P.W. *"Managing a Programming Project"*. Prentice-Hall, Englewood Cliffs, N.J., 1973.

[43] Milne, R.W. "A Few Problems with Expert Systems". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[44] Myers, Ware. "Introduction to Expert Systems". *IEEE Expert*, Vol. 1, No. 1, pp. 100-109, Spring 1986.

[45] Naedel, Dick. "Ada and Embedded AI". *Defense Electronics*, Vol. 18, No. 4, pp. 90-100, April 1986.

[46] Naughton, M., James, Dr. "A Major Conference on Expert Systems". In *Expert Knowledge Systems*, 1985. pp. 165-172.

[47] Orciuch, Edward and Gilbert, Kenneth and Marshall, Charles. "The Application of Modular Software Engineering Techniques to a Very Large Expert System". In *Proceedings of the Nineteenth Annual Hawaii International Conference*, January 1986.

[48] O'Reilly, C. "Application of AI, II". In *Proceedings of SPIE*, April 1985.

[49] Parnas, David Lorge. "Software Aspects of Strategic Defense Systems". *Communications of the ACM*, Vol. 28, No. 12, pp. 1326–1335, December 1985. SDI.

[50] Parsaye, K. "The Evolutionary Road to Expert Systems". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[51] Pearson, G. "Mission Planning within the Framework of the Blackboard Model". In *Proceedings of Expert Systems in Government Symposium*, October 1985.

[52] Polit, Stephen. "R1 and Beyond: AI Technology transfer at DEC". *The AI Magazine*, Vol. 6, No. 4, pp. 76–78, Winter 1985.

[53] Prerau, David S. "Selection of an Appropriate Domain for an Expert System". *The AI Magazine*, Vol. 5, No. 2, pp. 26-29, Summer 1985.

[54] Prerau, David S. "Selection of an Appropriate Domain for an Expert System". *AI Magazine*, Vol. 6, No. 2, pp. 26–30, Summer 1985.

[55] Reiner, Julius and Smith, Jeff. "Practical AI Impementation Issues in Real-Time Multisensor Function". In *Government Microcircuit Applications Conference (GOMAC)*, 1985.

[56] Royer, Thomas. "EIA Workshop Trip Report". September 1985. Personal Communication.

[57] Sandford, D. "*Parts I, II, III of KNOWLEDGE BASED LEARNING SYSTEMS DS + CVS – A Proposal for Research CVS = An Intro. to the Meta-Theory & Logical Foundations*". Technical Report, Rutgers University, Department of Computer Science, May 1980.

[58] Schwartz, Tom J. "MCC Says Better User Interfaces Key to Complex Systems". December 23 1985. Newspaper.

[59] Scown, Susan J. "The Artificial Intelligence Experience: An Introduction, Digital, Inc.". DEC Technical Report.

[60] Sheil, Beau. "Artificial Intelligence Tool Box". In *Artificial Intelligence Applications for Business*, 1984. Proceedings of the NYU Symposium Editor, Walter Reitman, May 1983.

[61] Sheil, Beau. "Power Tools for Programmers". *Datamation*, Vol. 29, No. 2, pp. 131-144, February 1983.

[62] Smith, Reid G. "On the Development of Commercial Expert Systems". *The AI Magazine*, Vol. 5, No. 3, pp. 61–73, Fall 1984.

[63] Stanley, Anne M. "B-1B Integrated Diagnostics, Proceedings from NSIA Conference in Alexandria". Unpublished paper.

[64] Stevenson, A. and Fox, M. and Rabin, M. "TESS: Tactical Expert System". April 1986.

[65] Tobat, D.L. and Rogers, S. K. and Cross, S.E. "SENTINEL: An Expert System Decision Aid for a Command, Control and Communication Operator". April 1986.

[66] Vessey, I. "Expertise in Debugging Computer Programs - A Process Analysis". *International Journal of Man Machine Studies*, Vol. 23, No. 5, pp. 459–494, November 1985.

[67] Waterman, Donald A. *"A Guide to Expert Systems"*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1986.

[68] Weiss, Sholom W. and Kulikowski, Casimir A. *"A Practical Guide to Designing Expert Systems"*. Rowman & Allanheld, Totowa, N.J., 1984.

[69] Winston, Patrick Henry. *"Artificial Intelligence 2nd Edition"*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1981.

[70] Yager, R. R. "Explanatory Models in Expert Systems". *International Journal of Man Machine*, Vol. 23, No. 5, pp. 539–550, November 1985.

Glossary

Ada - is a programming language that was designed to meet the needs of programmers and to embody the concept of design methodologies by encouraging and supporting good design and programming practices.

Artificial Intelligence - The science of making machines do tasks that would require intelligence if done by man. An approach that has its emphasis on symbolic processes for representing and manipulating knowledge in a problem solving mode.

Automatic Programming - The ability to use programs to automatically generate other programs.

Backward-Chaining - An inference method that begins with a goal and works backwards to seek a chain of premises that accounts for all the facts at hand.

Causality - Inference mechanism based on the understanding of the structure and/or function of a given device.

Computer Vision - Perception by a computer, based on visual sensory input, in which a symbolic description is developed of a scene depicted in an image. Used synonymously with image understanding and scene analysis.

Deduction - A process of reasoning in which the conclusion follows from the premises given.

Demon - A local rule or procedure which is triggered upon changes to specific properties in a structured knowledge base.

Domain - The problem area or region of knowledge(e.g. bacterial infections, fault diagnosis or computer configuration).

DWIM - Do What I Mean. Part of the InterLISP environment with user facilities such as correcting mispelled words, variables and code.

Empirical Association - Inference based on the association made from previous experience or observation.

Event Driven - A forward chaining, problem solving approach based on the current problem status.

Expert Systems - AI systems that reflect the skill, experience and judgement of humans knowledgeable in a particular field.

Explanation Facilities - Ability to provide a trace mapping on how a particular problem was solved.

Exploratory Programming - Conscious intertwining of system design and implementation.

Failure Tolerance - Term used in testing to statistically indicate the allowable failure rate. For example, to meet acceptance criteria for a given system, it must generate accurate results, say, 95 percent of the time.

Fault Diagnosis - Determining the trouble source in system.

Forward Chaining - An inference method that gathers pieces of information in an attempt to build forward to an end goal.

Frame-Based Knowledge Representation - A representation method that clusters closely associated knowledge about a class of objects or events. A frame is a data structure that associates one or more features with an object in terms of various slots and particular slot values. The slots may be filled by values or perhaps pointers to other frames.

Goal Driven - A problem solving approach that works backwards from the goal.

Heuristics - General rules of thumb used by an expert to process information in a particular problem area.

Inference Engine - A program's method of navigating through a knowledge base in an attempt to solve a problem.

Inferential Rule - An associated link between antecedent conditions and resultant beliefs that permits beliefs to be inferred from valid antecedent conditions.

Instantiation - Replacing a variable by an instance that satisfies the system or the statement in which the variable appears.

Intelligence - The degree to which an individual can successfully respond to new situations and problems. It is based on the individual's knowledge level and the ability to appropriately manipulate and reformulate that knowledge as required by the situation or problem.

Intelligent Assistant - An AI computer program (usually a knowledge-based system) that aids a person in the performance of a task.

Knowledge - Knowledge in AI is basically comprised of facts, beliefs and heuristic rules.

Knowledge Acquisition - The process of extracting and formalizing the knowledge of an expert for use by an expert system.

Knowledge Base - Collection of facts, inferences and procedures, corresponding to the types of information needed for problem solutions.

Knowledge Based Systems - Another name for expert system

Knowledge Engineer - A software engineer specializing in the techniques of knowledge acquisition and knowledge representation.

Knowledge Representation - Formalized structure of facts and heuristics that encompass the descriptions, relationships and procedures.

Knowledge Source - The body of domain knowledge which pertains to a specific problem. (i.e., expert).

LISP - A principal AI programming language. By definition, LISP, is highly recursive, and due to an untyped and applicative framework, highly supports symbolic computing. There are many LISP dialects which include:

- Common LISP;
- InterLISP;

- MacLISP; and
- ZetaLISP.

Metaknowledge - *Knowledge about the structure of* and how to use knowledge.

Metarule - A rule which prescribes the manner in which rules should be used.

Natural Language Understanding - The response by a computer based on the meaning of a natural language input.

Object Oriented Programming - A programming approach focused on objects which communicate by message passing. An object is considered to be a package of information and descriptions of procedures that can manipulate that information.

Planning Systems - An expert system application that performs planning design actions.

Predicate Calculus - An extension of propositional calculus. In predicate calculus, elementary units are called objects. Predicates are statements about objects.

Production Rule - A modular knowledge structure representing a single chunk of knowledge, usually in an If-Then or Antecedent-Consequent form.

PROLOG - An AI or symbolic programming language that is based on predicate calculus.

Prototype - An initial model or system that is used as a basis for building future systems.

Rapid Prototyping - An approach used in system development to quickly generate a working model that simulates a response to a simplified version of the problem at hand. Through incremental development, the simplified model is made increasing complex in ultimate response to the problem statement.

Robotics - One AI research branch that is involved with allowing computers to see and manipulate objects in a specific environment.

Rule-Based Knowledge Representation - A representation method comprised of a set of production rules based on a situation and an action...e.g. If Antecedent - then Consequent type of structure. A rule is a conditional statement consisting of a part comprised of one or more *if* clauses which establishes conditions that must apply if a second part, composed of one or more *then* clauses is to be acted upon.

Ruleset - A set of rules that constitutes a module of heuristic knowledge.

Satisfice - Achieve a solution that satisfies all imposing constraints.

Semantic - Specifies the meaning or significance of a symbolic expression.

Semantic Net - A knowledge representation scheme composed of nodes describing objects, and links describing relationships between nodes.

Shell - Software programs used to develop expert systems.

Slot - A description of an object in a frame. Slots may correspond to features such as name, definition or creator, or may represent a value.

Syntactic - Specifies the form or structure of symbol expressions.

Tools for Knowledge Engineering - Programming systems that facilitate the process of building expert systems. Particular emphasis is placed on generic task packages and very high-level languages for heuristic programming.

Truth Maintenance - The task of preserving consistent beliefs in a reasoning system whose beliefs change over time.

Acronyms

ABI/Inform	American Business Information
AFR	Air Force Regulation
AI	Artificial Intelligence
ART	Automated Reasoning Tool
ATO	Air Tactical Operations
ATP	Acceptance Test Plan
ATR	Automatic Target Recognition
BM/C^3	Battle Management/Command Control and Communications
CBD	Commerce Business Daily
CBTAO	Cost Benefit of Tactical Air Operations
CI	Configuration Item
CM	Configuration Management
CMS	Configuration Management System
CMU	Carnegie-Mellon University
COMPASS	Central Office Maintenance Printout Analysis and Suggest System
CPCI	Computer Program Configuration Item
CPL	Constraint Propagation Language
CPU	Central Processing Unit
CSCI	Computer Software Configuration Item
DARPA	Defense Advanced Research Projects Agency
DART	Duplex Army Radio/Radar Targeting
DATA	Decision Aids for Target Aggregation
DBMS	Data Base Management System
DEC	Digital Equipment Corporation
DID	Data Item Description
DLL	Data Layout Language
DOD	Department of Defense
DP	Data Processing
DPMA	Data Processing Management Association
DROLS	*Defense RDT&E On-Line System*
DSDS	Decision Support Development System
DTIC	Defense Technical Information Center
ECM	Electronic Countermeasures
ECS	Embedded Computer Systems
EIA	Electronic Industries Association
ESP	Expert Software Pricer
ESTAS	Expert System for Target Attack Sequencing
ETI	Expert Technologies, Inc.
FDRS	Fault Diagnosis and Resolution System
FRL	Frame Representation Language
HOL	High Order Language
IBM	International Business Machines Corporation
IBM FSG	International Business Machines Federal Systems Group

Acronyms

INSPEC	Information Services in Physics, Electrotechnology Computers and Control
IR&D	Internal Research and Development
IRAD	Internal Research and Development
ITACC	Integrated Tactical Air Control Center
IV&V	Independent Verification and Validation
JLC	Joint Logistics Commanders
KAR	Knowledge Acquisition Rules
KBS	Knowledge Based System
KBSA	Knowledge Based Software Acquisition
KEE	Knowledge Engineering Environment
KES	Knowledge Engineering System
LAS	Lockheed Aircraft Service Company
LES	Lockheed Expert System
LGC	Lockheed-Georgia Company
MIT	Massachusetts Institute of Technology
NASA	National Aeronautics and Space Administration
NSF	National Science Foundation
NTIS	National Technical Information Service
OS	Operating System
PDR	Preliminary Design Review
PDS	Problem Definition Statement
PGSC	PAR Government Systems Corporation
QA	Quality Assurance
RADC	Rome Air Development Center
R&D	Research & Development
SA&E	Software Architecture & Engineering, Inc.
SDI	Strategic Defense Initiative
SFAS	Sensitive Financial Analysis System
SPEA	See and Project Enemy Activity
SQR	System Quality Review
SSE	Software Systems Engineering Directorate
STAMP	System Testablility and Maintenance Program
TI	Texas Instruments, Inc.
TMS	Truth Maintenance System
UICP	Unidentified Command Post

Appendix A

Software Development Problems

This appendix isolates software related problems that surface in both conventional software development and in artificial intelligence systems development. The following software categorization was extracted from a report issued by the Department of Defense (DOD) Joint Services Task Force on Software Problems, *Report of the DOD Task Force on Software Problems*. One purpose of that study was to identify conventional software problems associated with embedded computer software.

1. Life Cycle:
   (a) Requirements;
   (b) Management;
   (c) Acquisition;
   (d) Product Assurance; and
   (e) Transition.
2. Environment:
   (a) Disciplined Methods;
   (b) Labor Intensive;
   (c) Tools;
   (d) Reinvention; and
   (e) Capital Investment.
3. Software Product:
   (a) Doesn't Meet the Need;
   (b) Software Metrics;
   (c) Design Attributes;
   (d) Documentation; and
   (e) Immutable Software.
4. People:
   (a) Skills;
   (b) Availability; and
   (c) Incentive.

Appendix A Software Development Problems

A short definition of each category and subcategory is provided followed by tables of conventional and AI systems problems that were identified in the following bibliographic sources:

1. Sources

   (a) Acquisition and Support of Embedded Computer System Software, September, 1981.

   (b) DOD Weapon Systems Software Management Study, June, 1975.

   (c) DOD Weapon Systems Software Acquisition and Management Study, Volumes I and II, May - June, 1975.

   (d) Final Report of the Joint Logistics Commanders' Workshop on Post Deployment Software Support (PDSS) for Mission Critical Computer Software, Volume II, June, 1975.

   (e) Proceedings of the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management, November 1, 1981.

   (f) Report of the DOD Task Force on Software Problems, Third Draft, July 15, 1982.

   (g) Report of the USAF Scientific Advisory Board, December, 1983.

   (h) Suggestions for DOD Management of Embedded Computer Software in an Environment of Rapidly Moving Technology, March, 1982.

   (i) *A Practical Guide to Designing Expert Systems*, Shalom M. Weiss and Casimir A. Kulikowski.

   (j) *Artificial Intelligence*, Patrick Henry Winston.

   (k) *Building Expert Systems*, Frederick Hayes-Roth, Donald A. Waterman, and Douglas B. Lenat.

   (l) "Computers and Information Technology in the Year 2000 - A Projection", Stephen F. Lundstrom and Ronald Lilarsen, IEEE, September 1985.

   (m) *Expert Systems*, Paul Harmon and David King.

   (n) "Expert Systems: Where Are We And Where Do We Go From Here?", Randall Davis, MIT AI Laboratory, June 1982.

   (o) *IEEE 1984 Workshop on Principles of Knowledge Based Systems*, IEEE Computer Society.

   (p) "On the Development of Commercial Expert Systems", Reid G. Smith, The AI Magazine, Fall 1984.

   (q) *Rule-Based Expert Systems*, Bruce G. Buchanan and Edward H. Shortliffe.

   (r) "Starting a Knowledge Engineering Project: A Step by Step Approach", Mike Freiling, Jim Alexander, Steve Messick, Steve Rehfuss, and Sherri Shulman.

   (s) *The AI Business*: The Commercial Uses of Artificial Intelligence, Patrick H. Winston and Karen A. Prendergast.

   (t) *The Artificial Intelligence Experience*, Susan J. Scown.

   (u) *The Handbook of Artificial Intelligence*, Volumes I-II, Avron Barr and Edward A. Feigenbaum.

A.1 Life Cycle

Life cycle refers to the relationship among activities of computer software development from its inception to retirement. The section is divided into five major categories: Requirements, Management, Acquisition, Product Assurance and Transition.

A.1.1 Requirements

The requirements phase of the software life cycle consists of the analysis and definition of a system as defined by the user. As such, communication between users, support agents and acquisition agents is essential to the success of the final system. Oftentimes, the requirement document does not accurately define the product; or perhaps uses ambiguous wording. Inadequate requirement information results in costly schedule changes to hardware and software. For itemized problems see tables A.1.6–1 and A.1.6–2.

A.1.2 Management

The management of software life cycle includes activities such as planning, monitoring of schedules and budgets, assigning responsibilities, tracking software projects and effective plans for software activities. For itemized problems see tables A.1.6–3 to A.1.6–5.

A.1.3 Acquisition

Software acquisition is difficult to specify because the rules for acquiring software have historically been based on the rules for obtaining hardware. Some major aspects of acquisition are:

- Management of System Interfaces - Good interfaces must be defined to guarantee a smooth integration;
- Project Status Reports - Documentation for project status;
- Documentation Requirements - Documentation where applicable;
- Software Change Control - Software must be carefully managed especially where changes occur;
- Reliance on Contractors - Several contractors are frequently responsible for managing the software process; and
- Design for the Complete Life Cycle - All resources needed: hardware, software, documentation, tools, personnel, etc. for the life cycle should be clearly identified.

For itemized problems see tables A.1.6–6 and A.1.6–7.

A.1.4 Product Assurance

Product Assurance involves demonstrating the correctness and quality of a software product at each level of the software life cycle. Higher quality software is achieved the earlier independent verification and validation of a system occurs in the life cycle. To provide a quality product, adequate resources for test planning along with organized communication between the acquisition agent, development contractor, test agent, support agent and user are essential. For itemized problems see tables A.1.6-8 and A.1.6-9.

A.1.5 Transition

Transition refers to the change in the software life cycle from one activity to another. Two examples involve the more difficult transitions from exploratory research to engineering development and from development to support (also known as maintenance). Problems occur because the transition from exploratory research to engineering development is characterized by new and rapidly changing technology, and contrarily, maintenance requires a stable technology base. For itemized problems see tables A.1.6-10 and A.1.6-11.

A.1.6 Life Cycle Problem Tables

The following tables show the problems which surfaced in each of the Life Cycle categories.

Table A.1.6–1: Conventional Software Requirements

	PROBLEM STATEMENT	SOURCE(S)
1.	Poor Communication Skills.	f
2.	Diverse interpretation of requirements.	c d f
3.	Requirements defined according to cost and schedule constraints.	b c f
4.	Inadequacy of English prose for software requirements.	c f h
5.	Excessive detail in requirements allocation.	b f
6.	System design characteristics appear in requirements documentation.	f
7.	Software is severely impacted by requirement changes.	a b c d e f g h
8.	Poor cost estimation analysis.	c d e f
9.	Hardware changes made with little knowledge of software implications.	b f
10.	Requirement changes are costly.	a b c d f g
11.	Lack of valid engineering data to define what is cost effective for support requirements.	f
12.	A support agent is not selected until late in process.	b f
13.	No operational concept developed.	b f
14.	Concurrency of the configuration of the trainer system does not always correspond to that of the primary weapon sytem.	f
15.	Protection of classified software.	f
16.	Security causes increased complexity due to the number of systems and levels of security required.	f
17.	Non-standard systems must be able to interface with others.	f
18.	Requirements are not concrete before development begins.	b c g
19.	Management budgets and schedules the software project without a true understanding of the requirements.	g
20.	There exists an inadequate understanding of the product to be developed.	b c g
21.	Systems continue to grow in complexity.	a b d e g

A.1.6 Life Cycle Problem Tables

Table A.1.6-2: Artificial Intelligence Requirements

PROBLEM STATEMENT	SOURCE(S)
1. Requirements oriented methods and intuitions learned in the development of other types of software do not carry over well to the knowledge engineering task.	r u
2. A major difficulty in the application of Artificial Intelligence (AI) systems is in obtaining a complete specification.	l
3. Problem Statements are inconsistent or incomplete in matching requirements to the final product.	k m o r s t u
4. Poor communication between engineers and experts.	t
5. Knowledge principles and problem-solving techniques are difficult to translate into a knowledge base to be used by an expert system.	l m
6. The knowledge is often ill-specified because the expert cannot always express exactly what he knows about his domain.	i n t u
7. When available at all, experts have little time at their disposal.	i
8. The knowledge acquisition phase is one of the most difficult and time consuming phases of expert system building.	i o u
9. Problem in determining the kind of knowledge required for an AI system.	i j k u

Overlapping AI References: 2(Management), 7(Design Attributes).

Table A.1.6-3: Conventional Software Management

	PROBLEM STATEMENT	SOURCE(S)
1.	Lack of skilled software program managers.	b f g h
2.	Air Force uses inexperienced, junior officers to direct programs.	f
3.	Misunderstanding of acquisition concepts.	b f
4.	Belief that hardware and software can be developed and supported by different concerns without strong engineering influence.	f
5.	Prototyping used to sell a system but does not achieve its goal of risk reduction.	f
6.	Incompatibilities between State Department and DOD decisions.	c f
7.	Lack of communication.	b c f
8.	Problem with the transfer of technological issues and the classification of data.	c f
9.	No clear assignment of responsibility and authority given to those working for management.	f
10.	Lack of useful data on projects making progress difficult to measure.	c f
11.	Lack of historical metrics and models.	b c d f
12.	Vague requirements definitions for complex systems.	c f
13.	Lack of tracking tools and planning.	c f
14.	Need for thorough software Development Plan to check validity.	c f
15.	Lack of performance monitors and modeling tools.	c f g
16.	Management budgets and schedules without a true understanding of the requirements.	g
17.	Often unrealistic budgets are drawn up just to win a contract proposal.	a b g
18.	Inaccuracies in software cost estimation models and techniques.	a b c d e g
19.	Contractual budget commitment is often made before Preliminary Design Review(PDR) occurs.	g
20.	Software is not reported properly in large system status reports.	b c g
21.	High risk software is not addressed in reports and/or reviews.	b c g

Continued on the following page

Table A.1.6–4: Conventional Software Management (Cont.)

PROBLEM STATEMENT	SOURCE(S)
22. Software status reports are too detailed to provide useful information or too costly to be prepared correctly.	b c g
23. Insufficient time for high level management to respond to potential problems indicated in status reports.	b c g
24. Good advice is sometimes ignored by program offices.	a
25. Cost information is rarely correlated with technical information for management purposes.	c
26. Management needs more information provided to identify where DOD software costs are occurring.	c
27. A high turnover rate of military software management personnel occurs in many program offices.	c
28. Very little knowledge available to acquisition managers for selecting the hardware, software, and firmware configuration items of a system.	e

Table A.1.6-5: Artificial Intelligence Management

		m	o	q	s	t	u
1.	Existing AI systems cover too narrow a range of expertise.						
2.	Lack of sufficient funding of AI projects exists.			r		t	
3.	Lack of strong management to guide an AI project through completion causes difficulties.					t	
4.	Do not know how to manage the transfer of progressive and evolutionary technology.		p				
5.	Knowledge engineering is often considered by management to be a technology that is far too difficult to even attempt.				r		
6.	There are difficulties in scaling from the current project sizes to large knowledge based systems.	l					

Overlapping AI References: 1(Requirements), 3(Requirements), 1(Acquisition), 1(Product Assurance), 1(Disciplined Methods), 1(Metrics), 2(Metrics), 1(Design Attributes), 7(Design Attributes), 1(Skills), 1(Availability).

A.1.6 *Life Cycle Problem Tables*

Table A.1.6–6: Conventional Software Acquisition

	PROBLEM STATEMENT	SOURCE(S)
1.	Many software contracts are inappropriate as they are designed for hardware.	f
2.	Changing requirements are not controlled.	f
3.	Software development tools are not required deliverables.	f
4.	No funds for software tools or documentation.	f
5.	Software cannot be carefully tracked or measured.	b c f h
6.	Documentation requirements are often starved for funds.	f
7.	Documentation is over-whelming in volume but marginal in value.	f h
8.	Requirements and design changes are not managed well, if at all.	b f
9.	Design changes are rarely made with the complete life cycle in mind.	a c f
10.	Decisions made with little consideration of cost over the life cycle.	a d f
11.	Even when software is not a primary cost or deliverable, it can have a large impact on system costs and schedules.	c
12.	Need to improve software acquisition management with more detail in early stages of development.	c

Table A.1.6–7: Artificial Intelligence Acquisition

PROBLEM STATEMENT	SOURCE(S)
1. The rapid prototyping method does not offer a clear guidance on how to produce a well-engineered commercial product.	p
2. With rapid prototyping, the system is always in a state of flux.	p
3. A precise set of tools needed to solve a problem cannot be identified before the implementation phase.	p
4. Representations at too specific or at too low a level yield problems in integration, maintenance and extensibility.	o

Overlapping AI References: 3(Requirements), 4(Management), 1(Disciplined Methods).

A.1.6 Life Cycle Problem Tables

Table A.1.6-8: Conventional Software Product Assurance

	PROBLEM STATEMENT	SOURCE(S)					
1.	Lack of criteria to determine how much testing is necessary at each stage of software development.	a				f	
2.	Testing is often governed by the time and money available.	a				f	
3.	Test tools are not a real measure of a products assurance.			c		f	
4.	The testing of some requirements is extremely difficult to simulate.					f	
5.	Software test planning does not receive an adequate share of total resources.	a				f	
6.	Incomplete or vague functional specifications cause errors discovered only during testing.	a	b			f	
7.	Software testing is not always provided throughout the entire program life cycle.	a		c		f	
8.	Independent Verification and Validation (IV&V) strategies are costly.					f	
9.	Errors exist in software after deployment to the user command regardless of the testing effect.				d		h
10.	There exists a high potential for a significant lapse in support during early deployment stages.		b				h
11.	Testing effort requires a significant amount of money.		b				

Table A.1.6-9: Artificial Intelligence Product Assurance

PROBLEM STATEMENT	SOURCE(S)
1. There exists a lack of knowledge regarding testing of AI systems.	o q t u
2. AI systems have no independent means of checking whether their conclusions are reasonable.	k
3. Experts should be evaluated as well as the systems.	q
4. Acquisition organization and structure of knowledge is a manual process with the requirement or detailed quality assurance.	l f

Overlapping AI References: 5(Requirements), 6(Requirements), 6(Design Attributes), 7(Design Attributes), 8(Design Attributes).

Table A.1.6-10: Conventional Software Transition

	PROBLEM STATEMENT	SOURCE(S)
1.	Rapid changes in technology.	a b d f
2.	Post engineering data becomes obsolete very quickly.	f
3.	Microprocessors and firmware are forced under software or hardware guidelines.	e f
4.	Need to develop a unique policy for microprocessors and firmware.	f
5.	Need for flexibility to adapt to technological change.	f
6.	Inability to develop and validate system and tactical software requirements and evaluate doctrinal problems.	f
7.	Insufficient acceptance testing performance on a regular basis for automated systems.	c f
8.	Necessary to develop and support automation.	f
9.	Need for a single, integrated set of procedures, guidelines, and standards for development of automated systems.	c f
10.	Lack of identification and use of an effective procedure for system development.	b c f
11.	Automated Systems have poorly defined function and interface requirements specifications.	d f
12.	Automated Systems lack proper modularization, use machine-orientated languages, and are inadequately documented.	f
13.	Most Research & Development (R&D) work is not aimed at a specific weapon system.	f
14.	Rarely is the transition made from the useful techniques and tools of R&D to operational systems.	f
15.	Post Deployment Software Support has to reorient its work force from traditional logistics and maintenance functions to advanced technology support.	a b g
16.	Lack of discipline in software development methodology.	b c h
17.	Delays in delivery of software has a large impact on system costs and schedules.	c
18.	Need to educate involved organizations before useful output is obtained.	c
19.	Need a specification or set of criteria to aid in the documentation selection process.	e

Table A.1.6–11: Artificial Intelligence Transition

PROBLEM STATEMENT	SOURCE(S)
1. Expansion of technology in the next five years will be explosive.	i s u
2. Terminology is inconsistent in the AI world.	k m
3. Systems only evolve gradually because it takes a good deal of experimentation to achieve high performance.	k
4. AI systems require a continuous relationship with an expert.	t
5. Eventually, as a result of AI systems, automation will occur and many jobs will disappear or change radically.	m
6. No programs exist yet which can understand simple, mechanical causality.	s
7. AI systems are unable to recognize or deal with problems for which their own knowledge is inapplicable.	k
8. AI methods must be augmented with conventional techniques to solve real problems.	p

Overlapping AI References: 1(Management), 4(Management), 1(Acquisition), 2(Acquisition), 3(Acquisition), 1(Tools), 9(Design Attributes), 1(Skills), 1(Availability).

A.2 Environment

The section on environment examines tools and methodologies involved in the development and in-service support of computer software. Five major categories which have surfaced are described: Disciplined Methods, Labor-intensive, Tools, Reinvention and Capital Investment.

A.2.1 Disciplined Methods

Disciplined methods refers to the use of adequate engineering discipline in software activities. Lack of software discipline varies from those who fail to apply the required discipline to a software problem, to those who fail to recognize the need for software discipline. The result is lack of enforcement of good programming standards which includes effective use of structured programming techniques, inefficient software configuration management, lack of baselining and documentation, and an insufficient system engineering technique applied to a program. For itemized problems see tables A.2.6-12 and A.2.6-13.

A.2.2 Labor Intensive

Since software is a labor-intensive technology, methods to improve the efficiency of people are a continuous concern. One means to increase productivity is the automation of manual processes. Another suggested means, is not only to reduce mechanical human activities, but to enhance creative possibilities through automated systems. For itemized problems see tables A.2.6-14 and A.2.6-15.

A.2.3 Tools

Development and support tools are needed to improve productivity in design, coding, test, support and management. Due to the diversity and complexity of systems, emphasis is placed on tools and procedures with wide-spread application. The needs for automated software design and support tools include uniform, portable, yet tailored tools with applications towards each phase of the software life cycle. Other desirable tools include: software documentation systems; configuration management systems; data base management systems; management information systems; software libraries; and comprehensive software development, support and test tools (editors, code syntax checkers, scenario generators, operational test data reduction software, etc.). For itemized problems see tables A.2.6-16 and A.2.6-17.

A.2.4 Reinvention

Reinvention refers to the inability of software development to reuse functionally similar software developed for other systems resulting in higher development costs. For itemized problems see tables A.2.6-18 and A.2.6-19.

A.2.5 Capital Investment

Capital investment is required to solve many existing software problems. Yet, software is given a lower priority in funding by the DOD. The funds allocated for capital investment and labor are signficantly lower than the estimated need. For example, there is an unwillingness to make the essential capital investment required to provide better support environments with an ultimate goal of improving software engineering and eliminating reinvention. For itemized problems see tables A.2.6–20 and A.2.6–21.

A.2.6 Environment Problem Tables

The following tables show the problems which surfaced in each of the Environment categories.

A.2.6 Environment Problem Tables

Table A.2.6-12: Conventional Software Disciplined Methods

PROBLEM STATEMENT	SOURCE(S)
1. Retraining of individuals is costly so movement is not promoted.	f
2. Contractor's software tools rarely leave their shops when the product is delivered.	f
3. Lack of consistent, disciplined methods impact Embedded Computer Systems (ECS) because the software is developed by independent groups.	f
4. Need for better definitions of software terms, measures of software qualities, and methods of measuring them.	c d h
5. Differing policies exist among various Federal agencies and industry.	h
6. Lack of standardization of Data Item Descriptions.	h
7. Within DOD, partial and inadequate standards exist.	h
8. Standards are rigid and lack the ability to be tailored.	b h
9. Lack of well-defined, consistent requirements causes Software Quality Assurance difficulties.	c h
10. No standard set of software acceptance criteria within DOD.	b c h
11. Government standards do not recognize that differences in program size should influence the decision to apply standards.	h
12. Standards address too much detail disregarding the end product objectives.	h

Table A.2.6–13: Artificial Intelligence Disciplined Methods

PROBLEM STATEMENT	SOURCE(S)
1. Methodologies for developing expert systems by extracting, representing, and manipulating an expert's knowledge have been slow in coming.	r

Overlapping AI References: 2(Transition), 1(Metrics), 2(Metrics).

A.2.6 Environment Problem Tables

Table A.2.6–14: Conventional Software Labor Intensive

PROBLEM STATEMENT	SOURCE(S)
1. Inefficient use of people in software development.	f
2. Software is a labor-intensive technology.	f g
3. Inadequate information support for tracking and control.	g

Table A.2.6-15: Artificial Intelligence Labor Intensive

PROBLEM STATEMENT	SOURCE(S)
1. Acquisition, organization and structuring of knowledge is a manual process.	I

Overlapping AI References: 1(Skills), 1(Availability).

A.2.6 Environment Problem Tables

Table A.2.6–16: Conventional Software Tools

	PROBLEM STATEMENT	SOURCE(S)				
1.	Contractor is often forced to use a government tool which requires use of an unfamiliar system.			f		
2.	Need for standardization of high order programming languages.			f		
3.	Deviations in language definition and implementation forbids transportability of High Order Language (HOL) programs.			f		
4.	Lack of consistent standards for software causes problems in acquisition and development.			f		h
5.	Effective and efficient maintenance is difficult.			f		
6.	Software is often re-engineered.	a		f	g	
7.	Tools are unavailable, unpublicized, difficult to understand or use, or inefficient.	a	c	f	g	h
8.	Inconsistent computer support of software development.		c	f		
9.	Difficulties in transitioning generic (non-weapon) specific software tools to weapon system programs.			f		
10.	Non-modular tools are: • expensive; • untimely; • difficult to maintain; • inflexible to change; • unreliable; and • non-responsive to user requirements.			f		
11.	[illegible] and host/target processors.					
12.	Resource constraints only allow a minimal number of tools to be developed.			f		
13.	Poor documentation of tools results in a lower level of quality results of the system.			f		
14.	Non-uniformity among tools.			f		
15.	Strive for language independence rather than support for a specific language.			f		
16.	Should apply existing tools.		c		g	

Table A.2.6-17: Artificial Intelligence Tools

PROBLEM STATEMENT	SOURCE(S)
1. Cost performance standards are not fully understood yet.	m

A.2.6 Environment Problem Tables

Table A.2.6-18: Conventional Software Reinvention

PROBLEM STATEMENT	SOURCE(S)
1. Designs and implementations of previous systems are not captured and reused in succeeding systems.	a b c d e f g h
2. A rigorous interface standard must be provided for all candidate reusable components.	f
3. A library of reusable components must evolve for use by all development activities.	f
4. An index of reusable components must be included.	b f
5. A rigorous design standard must be provided for any major software system or subsystem to be implemented as a skeleton.	e f
6. A library of subsystem or system skeletons must be provided.	f
7. A set of supporting tools must be implemented.	f
8. Most application-specific software is developed new for each application.	g

Table A.2.6-19: Artificial Intelligence Reinvention

PROBLEM STATEMENT	SOURCE(S)
99. None Noted	

Overlapping AI References: 2(Acquisition), 4(Transition), 6(Transition), 8(Transition).

A.2.6 Environment Problem Tables

Table A.2.6-20: Conventional Software Capital Investment

PROBLEM STATEMENT	SOURCE(S)
1. Many DOD Computer Support facilities are overloaded and use aging equipment.	f
2. Software is developed on target hardware which is not designed to support such development.	f
3. Lack of sufficient time is spent on Independent Research and Development.	f
4. Schedule slips occur resulting in cost and resource consumption.	b
5. Major software costs in development, operation, and maintenance phases.	c

Table A.2.6-21: Artificial Intelligence Capital Investment

PROBLEM STATEMENT	SOURCE(S)
99. None Noted	

Overlapping AI References: 2(Management).

A.3 Software Product

The software product consists of the operational embedded computer software as well as the materials necessary for life cycle support - requirement and design specifications, source code, test data, system generation data, unique support tools etc. The following five categories briefly describe some subdivisions within the Software Product: Doesn't Meet the Need; Software Metrics; Design Attributes; Documentation and Immutable Software,

A.3.1 Doesn't Meet the Need

The section refers to the failure of the deployed system to meet the user's need. One such situation occurs when requirements are either stated or implemented incorrectly which results in changes to correct the problems. For itemized problems see tables A.3.6–22 and A.3.6–23.

A.3.2 Software Metrics

Software metrics are aimed at providing analytic models and empirical data on software to aid in the choice of software engineering techniques, estimate development resources, and evaluate future costs. For itemized problems see tables A.3.6–24 and A.3.6–25.

A.3.3 Design Attributes

The Design provides an acceptable programming solution to the problems stipulated in the Requirements Specification. As such, the design specification contains a solution to the user's problem. For itemized problems see tables A.3.6–26 to A.3.6–28.

A.3.4 Documentation

Documentation should convey information on or about the computer system developed. Both managerial and technical work should be included as well as transitional documentation helpful in bridging the gap between phases. Also, documentation serves as a baseline from which changes or upgrades are made. Although documentation is an important aspect in the development of computer systems, the resources allocated do not reflect the actual costs necessary to produce adequate documentation. For itemized problems see tables A.3.6–29 and A.3.6–30.

A.3.5 Immutable Software

This section refers to software packages that are system unique, non-portable and non-reusable. Software developed for embedded computer systems is generally tailored to the specific application and hardware environment, without regard for reusability. Consequently, acquisition agencies are forced to pay over and over again for software that has essentially been written elsewhere. For itemized problems see tables A.3.6–31 and A.3.6–32.

A.3.6 Software Product Problem Tables

The following tables show the problems which surfaced in each of the Software Product categories.

A.3.6 Software Product Problem Tables

Table A.3.6-22: Conventional Software Doesn't Meet the Need

PROBLEM STATEMENT	SOURCE(S)
1. Software is expected to solve hardware problems.	b f
2. Software is difficult to develop and support, understand completely, and measure its performance fairly.	a b c d e f g h
3. Inadequate communication in system.	c f
4. Ambiguous, unclear, and incomplete requirements definition.	b c f
5. Necessary to define completion and performance criteria.	f
6. Incompatibilities between test equipment at different levels of maintenance.	f

Table A.3.6-23: Artificial Intelligence Doesn't Meet the Need

PROBLEM STATEMENT	SOURCE(S)
1. Myths regarding human and computer intelligence may mislead the user in terms of system capabilities.	j

Overlapping AI References: 6(Requirements), 9(Requirements), 1(Product Assurance).

A.3.6 Software Product Problem Tables

Table A.3.6-24: Conventional Software Metrics

PROBLEM STATEMENT	SOURCE(S)
1. Lack of good analytical methods and hard empirical data needed to estimate future costs and mission impacts.	a b c d e f
2. No validated models of life cycle costs and productivities in development and support.	a b c e f
3. Unexpected changes in hardwired software during software project life cycle.	f
4. Undecided as to whether or not Configuration Item (CI) or Computer Program Configuration Item (CPCI) standards should be applied to firmware.	e f
5. Need to separate and conduct measurement of creativity and implementation.	f
6. Contractors are not requested to report detailed software data on any aspect of software acquisition or support.	a
7. Managers need regularly scheduled updates to show cost and schedule states.	d e

Table A.3.6-25: Artificial Intelligence Metrics

PROBLEM STATEMENT	SOURCE(S)	
1. Evaluation of an AI project is difficult.	q	u
2. There exists no true scale of measure to tell how high a rating that a system can realistically achieve.	q	

Overlapping AI References: 4(Management), 1(Acquisition), 2(Acquistion).

A.3.6 Software Product Problem Tables

Table A.3.6-26: Conventional Software Design Attributes

	PROBLEM STATEMENT	SOURCE(S)
1.	Need a more focused body of literature for software design as it relates to system computational architecture and performance.	a f
2.	Lack of an understanding of both software design implications and hardware architecture.	b f
3.	Need for safety features to prevent loss of security information.	f
4.	Design requirements do not exist to ensure the best user system interface.	b f
5.	Misunderstandings occur between software and hardware engineers.	f
6.	Software engineers develop misconceptions of how hardware performs.	b f g
7.	Inadequately designed requirements.	b c f g
8.	Incorrect assumptions made by software engineers.	f g
9.	Software often lacks modularity resulting in memory waste.	b f
10.	Improper selection of a software language to be used.	b f
11.	Programming style contributes to faulty design.	b f
12.	Poor documentation and conformity to a standard.	a f
13.	Software is often written to fit the outdated hardware system used.	a f
14.	Too ambiguous in defining objectives.	f
15.	Monolithic development leads to cost overruns and schedule delays.	f
16.	The software life cycle is not really considered when design decisions are made.	f
17.	Unclear understanding of design options.	c g
18.	It is often the case that in order to include a new capability in a system a major software redesign is required.	a b c d e f g h
19.	Premature programming begins causing long-term difficulties.	b
20.	Bottom-up design is implemented rather than top-down.	b
21.	Long, costly design phase.	b

Continued on the following page

Table A.3.6-27: Conventional Software Design Attributes (Cont.)

PROBLEM STATEMENT		SOURCE(S)
22.	Lack of software transferability from one system to another.	b c d
23.	Software does not have the same degree of visibility or attention as hardware does.	c
24.	The major elements of weapons system software are often not integral with the operational components.	c
25.	Weapons system software does not fit previously defined procurement categories.	c
26.	Need to allocate software resources differently than hardware portions of a system program.	c
27.	Known risk reduction techniques need to be employed for software.	c
28.	Need simplicity of interfaces among CI's and CPCI's.	e

A.3.6 Software Product Problem Tables

Table A.3.6-28: Artificial Intelligence Design Attributes

PROBLEM STATEMENT	SOURCE(S)
1. Problems arise in making associative data bases (which require functions for adding, removing and fetching data items) efficient and meaningful.	j
2. AI systems are not noted for their speed.	m o s t
3. In AI programs, data structures tend to be large and complex.	n
4. Improving and defining descriptions often mobilizes powerful constraints that force conclusions directly without sophisticated problem solving and reasoning mechanisms.	j
5. Simple search techniques cannot find optimal paths and may be inefficient.	j
6. Difficulties arise when the expert attempts to map his explanations directly into the formalism.	p
7. There often exists voluminous amounts of information in the definition phase of an expert system.	i r s t u
8. Experts often trim their knowledge to fit the knowledge structure more conveniently.	t
9. Knowledge bases exist but little help is available for initial design decisions.	i k
10. Because knowledge systems are not hierarchical, they are difficult to decompose.	o

Overlapping AI References: 3(Requirements), 4(Requirements), 5(Requirements), 6(Requirements), 9(Requirements), 4(Management), 1(Acquisition), 3(Acquisition), 2(Transition), 4(Transition), 6(Transition), 8(Transition), 1(Incentives).

Table A.3.6–29: Conventional Software Documentation

	PROBLEM STATEMENT	SOURCE(S)
1.	Documentation effort is relaxed when schedule slips occur.	a b d f
2.	Documentation is often deleted if funding is reduced.	a b d f
3.	There exists inconsistencies as to what documentation is necessary.	a d e f
4.	Errors occur in the actual production of documents.	a f
5.	There exists little or no documentation in key areas of design and analysis.	a b d f
6.	Inadequate documentation exists resulting in a lack of complete project history.	a f
7.	Requirements of information scope, depth, and format are usually ignored.	a
8.	In contracting for software documentation, there exist problems in knowing what to ask for.	a e
9.	No standard documentation Data Item Description's (DID's) for software.	a
10.	Factors of difficulty with regard to documentation: • system complexity; • project resources; • development stage; and • documentation overlap.	e
11.	System interfacing must appear in documentation.	e
12.	Need sufficient manpower to manage/review the documentation.	d e
13.	Anticipated maintenance requirements must be developed.	d e
14.	Traceability between documents must exist.	e
15.	Problem in defining what comprises a hardware intensive versus software intensive application.	e

A.3.6 Software Product Problem Tables

Table A.3.6-30: Artificial Intelligence Documentation

PROBLEM STATEMENT	SOURCE(S)
99. None Noted	

Table A.3.6-31: Conventional Software Immutable Software

PROBLEM STATEMENT	SOURCE(S)		
1. No formal approach exists to exploit the potential of reactions to adversary threats within systems.			f
2. No emphasis on systems to meet changes in enemy defensive techniques.			f
3. Designers do not optimize available computer resources.	b	c	f
4. Lack of standard hardware used.			f
5. Inadequate transfer of information regarding existing software resources.			f
6. Specifications are not written in a common language or format.	b		f
7. Software is not documented to be reused.			f
8. Lack of software development standards exist.	b		f

A.3.6 Software Product Problem Tables

Table A.3.6-32: Artificial Intelligence Immutable Software

PROBLEM STATEMENT	SOURCE(S)
99 None Noted	

Overlapping AI References: 4(Management), 1(Acquisition).

A.4 People

A shortage of skilled system engineers, software engineers and managers has surfaced as a result of the rapid spread of digital technology. Consequently, three problematic areas have manifested themselves in professional skills, professional availability, and professional incentive which are examined as separate categories in this section.

A.4.1 Skills

One of the most highly valued and respected skills of an engineer is the ability to properly prepare software requirements, designs, and to provide adequate support for future modifications to systems. For itemized problems see tables A.4.4-33 and A.4.4-34.

A.4.2 Availability

The availability of qualified professionals for software development does not meet the need. For itemized problems see tables A.4.4-35 and A.4.4-36.

A.4.3 Incentive

Incentive refers to the idea that the software engineering career field should recognize and encourage skilled professionals through incentives such as better working conditions, educational opportunities, promotions and salary increases. Along with this concept is the idea that professionals should be trained in managerial and technical skills. For itemized problems see tables A.4.4-37 and A.4.4-38.

A.4.4 People Problem Tables

The following tables show the problems which surfaced in each of the People categories.

A.4.4 People Problem Tables

Table A.4.4-33: Conventional Software Skills

PROBLEM STATEMENT	SOURCE(S)
1. Shortage of skilled systems engineers, software engineers, and managers.	a f g
2. Individuals must require a broad range of knowledge in many areas.	f
3. Communication techniques are often ineffective.	f
4. Managers often lack technical knowledge.	b f
5. No formal instruction adequate to develop needed skills.	f
6. A few scarce experts must guide an entire software project.	d g

Table A.4.4-34: Artificial Intelligence Skills

PROBLEM STATEMENT	SOURCE(S)
1. Need for multiple experienced managers and engineers in the field of AI.	n

Overlapping AI References: 4(Requirements), 6(Requirements), 3(Management), 1(Product Assurance), 1(Availability).

A.4.4 People Problem Tables

Table A.4.4-35: Conventional Software Availability

PROBLEM STATEMENT	SOURCE(S)
1. Greater demand for engineers and computer scientists than individuals actually exist.	a f
2. Number of people actually needed on a software project is not always clear.	f
3. Lack of experienced technical personnel within the Navy.	f
4. Severe personnel shortage.	f g

Table A.4.4-36: Artificial Intelligence Availability

PROBLEM STATEMENT	SOURCE(S)
1. There only exists a limited number of engineers in AI software development.	i k o p s t u

Overlapping AI References: 5(Requirements), 7(Requirements), 1(Skills).

A.4.4 People Problem Tables

Table A.4.4-37: Conventional Software Incentive

PROBLEM STATEMENT	SOURCE(S)	
1. Sellers market exists; high rate of turnover.		f
2. Excellent technical people are promoted out of their fields into management positions.		f
3. Lack of reward for excellence.		f
4. Industry outbids the Government for desirable candidates.	a	f
5. Standard three year tour of duty policy exists for Air Force officers.	a	

Table A.4.4-38: Artificial Intelligence Incentive

PROBLEM STATEMENT	SOURCE(S)
1. To some individuals, the development of AI systems is perceived as a direct threat to their own jobs and security.	i
2. No existing non-industrial environments where young researchers can work for five or 10 years with good equipment.	s

Overlapping AI References: 1(Transition).

Appendix B

Questionnaire

ARTIFICIAL INTELLIGENCE SOFTWARE DEVELOPMENT CASE STUDY QUESTIONNAIRE

The attached questionnaire has been sent to you by the Software Systems Engineering Directorate (SSE) of Sanders Associates Inc., a large defense electronics firm located in southern New Hampshire. SSE provides software support to the users of Sanders computing facilities and undertakes a broad spectrum of software engineering projects.

One such project requires SSE, under contract to Rome Air Development Center[1], to perform a study on the acquisition, management, and control of AI software. While the Department of Defense has established numerous standards, policies, and guidelines for the acquisition and development of many types of conventional software, no such information exists for AI software. The primary purpose of this study is to postulate a model for the development of AI software which could subsequently be used to devise associated policy, standards, and guidelines. One way we propose to address our contract requirements, is to perform a number of case studies on AI system development. In this way, we can attempt to ferret out common methodologies and pay particular attention to those techniques that are considered to be successful. Consequently, the objective of the attached questionnaire is to obtain data from sources active in the AI arena to support the development of a viable model.

The questionnaire is divided into four parts. The first part provides an introduction to the overall questionnaire by discussing the concept of a software development model. One model typically used by the Department of Defense is presented pictorially and discussed briefly. The second part solicits information of a background nature that may be used to weigh, at least in a qualitative sense, responses to the questions in the remaining two parts. The third part is devoted to technical issues regarding the overall construction and support of the system. This part is further divided into five sections. The first section contains general questions pertinent to modeling AI software development. The remaining four sections parallel, in a broad sense, major activities associated with conventional software development and field support that, we assume, have counterparts in AI software development. Finally, the fourth part contains miscellaneous questions whose placement in one of the sections of Part III would have been inappropriately narrow.

It is intended that the response to the questionnaire be devoted to experiences encountered with one AI system. If you've had experience with more than one system, perhaps you could choose the one that you think best describes the AI software development process. We suspect that when you review the questionnaire you will find that many questions can be answered quite briefly while others will require more lengthy responses. In order to reduce the amount of time you might spend in

[1] Contract No. F30602-85-C-0254, Technical Contact: Richard Evans, 315-330-3528

responding to the more demanding questions, you may attach or reference available documentation where appropriate. We would appreciate any references be as specific as possible.

We greatly appreciate your important input to this effort and will gladly acknowledge your contribution in our Final Report. Furthermore, we will be happy to share any nonconfidential information that we collect from other sources.

If you have any questions or comments concerning this questionnaire, please feel free to contact us as indicated below.

Again, thank you for your cooperation.

Sandy King
Sr. Software Engineer
603-885-9242

Larry Fry
Study Director
603-885-9208

Sanders Associates
Software Systems Engineering
MER24-1283
95 Canal Street
Nashua, NH 03061

PART I: INTRODUCTION

Systems developed for the Department of Defense (DOD) are becoming increasingly complex, requiring more time and a larger budget to complete. From a historical standpoint, DOD experience indicates a number of cases where the acquisition of software has exceeded the predetermined cost and schedule. To assure a higher success rate in acquiring systems, the DOD has set several objectives. Principle objectives among them are that the contracted system:

- satisfy the needs defined by the DOD, and
- be developed within the allocated budget and schedule.

Consequently, the DOD has mandated that contractors adhere to a methodical engineering approach in order to provide visibility and control over the development process.

The engineering approach mandated by the DOD requires various levels of system and software definition to occur over time. At the completion of each level of definition the products (i.e. documentation or code) associated with that level are *frozen in a state* such that any future changes must be treated in a formal manner. These *baselines,* as the DOD calls them, consist of *functional, allocated, and product* baselines (see Figure 1). The functional baseline defines requirements for the overall system. The allocated baseline freezes software requirements and the product baseline reflects the "as built" design. A formal review, attended by representatives of the government and the contractor, is the vehicle for ascertaining readiness for establishing a baseline. Once established, these baselines are under government control and any changes thereto must be agreed to by the government prior to implementation. Representative of the controlled baseline entity is a document in which information appropriate to the baseline is recorded.

In addition to the government controlled baselines, the most recently issued DOD software development standard, DOD-STD-2167, also requires that the developer, through the *developmental configuration,* control the design and the code as it evolves. Inherent in this stipulation is the requirement to conduct numerous internal reviews which are intended to act as checkpoints for determining development progress.

Because DOD systems exist in a constantly changing environment, planning and designing for change is a high priority requirement within the DOD. Standards such as DOD-STD-2167 address this requirement by stipulating analysis, design, and coding standards to be adhered to by contractors. These standards are aimed at producing software that, when fielded, can be easily maintained and modified.

Because of its research oriented nature, the development of AI software has not reached the maturity of process/product standardization that conventional software has achieved. The DOD recognizes the importance and value of AI software and is concerned that its development be managed and controlled in a manner which will provide visibility into its development. From the perspective of the DOD, the desire to acquire a high quality, maintainable product applies equally to AI software as it does to conventional software.

With this background in mind, the questionnaire is oriented towards obtaining data that will highlight the activities involved in AI software development, the approach taken, and, just as importantly, the management visibility and control over the product as it evolves. Although the

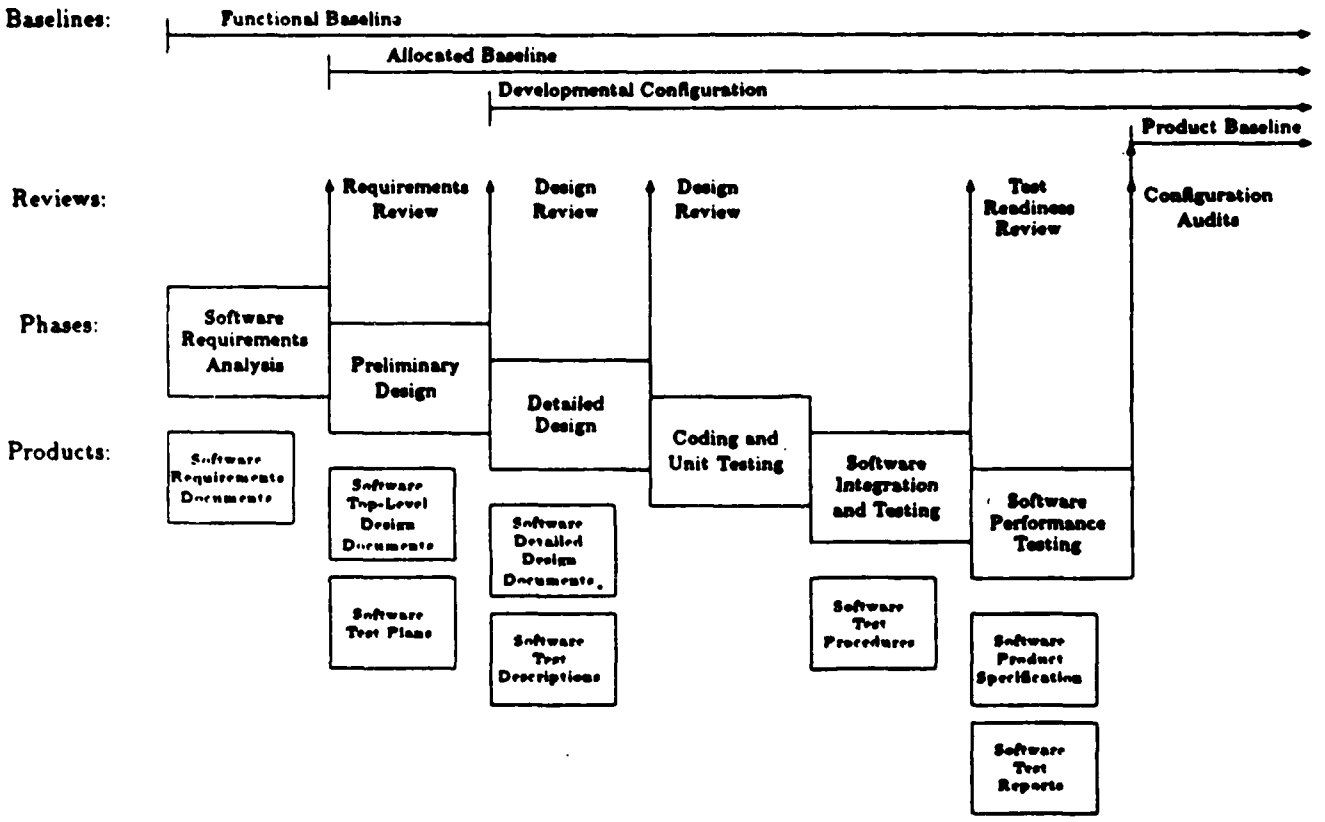


Figure B-1: Software Development Cycle

process of AI software development may vary significantly from the process presented in conventional software development models, the need for managerial insight must be satisfied. For instance, systems which take three to four years to develop must be measurable in terms of progress and expectations. To this end, we hope the questions will provide the data to develop models which accurately represent the technological and management issues involved in AI software development.

PART II: BACKGROUND

The purpose of this part is to identify basic introductory information about the subject system. In addition, certain information is requested to help put in proper perspective the responses to engineering and management related questions contained in Part III of this questionnaire.

B1. Please identify the name of the system to which the responses to this questionnaire are applicable. Briefly describe the purpose and nature of the system and how someone may use the system to solve problems relevant to the system purpose.

B2. Was the source(s) of funding for the project internal, contractual, or a grant? If other, please specify.

B3. Was this the first AI project that your company was involved in? If not, how many systems did the company develop prior to this one?

B4. How many engineers were assigned to the project? Of these, how many had AI development experience prior to this project?

B5. How long did it take to develop the system (man-months) from problem definition to first prototype? How much longer (after the first prototype) did it take to field the system? Was the schedule predefined either by management, the customer or the amount of funding allocated for the entire project?

B6. How many engineers assigned to the project had prior experience in the development of conventional software for the Department of Defense? Were military standards dictated on any of these projects? To what extent would you say that military standards, or any other standards governing the development of conventional software (please identify), influenced the development of the AI system?

B7. The success of a system can be measured by many criteria. Some criteria may be:

- demonstrating the feasibility of a new technology,
- overall user satisfaction,
- significant productivity gains

To what degree do you consider the system a success for each of the above criteria? For those criteria for which you weren't successful, is there something you would do differently in your next endeavor to improve your overall success? If so, what would that be? Are there any other criteria against which you would measure the success of the system? If so, what are they and how did your system fare against these criteria?

PART III: DEVELOPMENT CYCLE

The purpose of this part is to determine the detailed technical and management issues that influenced the scope, development, and acceptance or certification of the system. This part is divided into five sections: general procedure, requirements definition/analysis, system construction, system evaluation/validation and field support. Note that the division is arbitrary and does not imply that AI software should necessarily be developed in a framework consisting of five corresponding phases. However, the phases do parallel major objectives in developing good conventional software and may have some applicability, in a broad sense, to the development of good AI software.

GENERAL PROCEDURE

GP1. A model for AI software development could consist of the following stages:

- Identification - determining problem characteristics
- Conceptualization - finding concepts to represent knowledge
- Formalization - designing structures to organize knowledge (tool building)
- Implementation - capturing knowledge
- Testing - validating a knowledge base and system behavior

Please comment on the suitability of the above stages for a *high level* (i.e., general) AI software development model. Also, please contrast the above steps with the approach or steps that were followed in the development of the subject system. Indicate whether the steps were followed in a sequential manner or iterated upon some number of times. Lastly, please provide a brief description of any products (i.e. documentation or code) generated as the result of each step.

GP2. Conventional software development efforts typically include review sessions, such as design reviews and code walkthroughs, to track interim project progress. These review sessions may be formal (customer attended) or informal (internal staff only). If applicable, please describe any review sessions held and/or any documentation produced during the development effort to ascertain progress.

REQUIREMENTS DEFINITION/ANALYSIS

RA1. Conventional software development efforts typically begin with a requirements analysis phase to identify and bound the problem at hand. Did you perform a requirements analysis for the subject system?

RA2. If the answer to RA1 is "yes", proceed to RA3. If not, how did you bound the goals of the development effort?

RA3. If the answer to RA1 is "no", proceed to the next section. Otherwise, describe the activities associated with the requirements analysis phase.

1. Were users included in the requirements analysis process? If so, to what extent was their input useful in defining the system?
2. Were any software tools used in the requirements analysis?
3. If documentation was generated, describe it. Was the documentation reviewed for approval?
4. Were the requirements frozen at the end of the requirements analysis such that any future changes had to go through an approval process?

SYSTEM CONSTRUCTION

SC1. In formulating a knowledge base, the knowledge acquisition process may be exercised in a number of ways. For example, one approach is for a knowledge engineer to study the domain area and then interrogate an expert. The interrogation may be tape recorded to provide a verbatim transcript for further reference and review. The information obtained in the interview ma, be assembled and ordered into subcategories from which a knowledge representation approach can be determined. Describe the knowledge acquisition process for your project. Include whether or not the process was documented. If so, was the documentation reviewed for approval? Lastly, what determined completion of the knowledge acquisition process?

SC2. Some methods of knowledge representation include conceptual dependencies, semantic networks, frame based, logic based, and production rule structures. What method(s) was selected for your system? If applicable, how was the method reviewed and approved? Also, please discuss the means employed to accommodate ill-defined or incomplete knowledge.

SC3. In a rapidly developing discipline the domain knowledge may be continually increasing. Since initially formulated, has the size of your knowledge base (number of rules, frames, etc.) expanded? If it has, was it necessary to obtain approval for each addition to the knowledge base? If so, describe the approval mechanism. Lastly, was there a point at which it was decided not to include any additional knowledge in the knowledge base, and was this a managerial or a technical decision?

SC4. The type of knowledge representation method selected often determines the programming tools used. What tool(s) was selected for inclusion in the subject system? If applicable, include a description of conventional languages and components used and how they fit into the overall system.

SC5. (Expert Systems Only) What dimensions were considered in identifying the "expert" behind the expert system? Was there any attempt made to verify the accuracy of his or her input prior to the system validation phase?

SC6. (Expert Systems Only) In developing a knowledge base for an expert system, input from more than one expert may result in a mixed approach that is not representative of any individual expert. How many experts had input to your knowledge base? If more than one, how was the information handled to eliminate conflicting and/or redundant facts and approaches?

SC7. Were users introduced into the development process? If so, to what extent did user participation influence the evolution of the developing system?

SC8. Exploratory programming is the conscious intertwining of system design and implementation. Namely, as the system is implemented, design changes may be warranted. Did your system go through an iterative period of design and implementation changes? If so, what was the magnitude of the changes and was it necessary to obtain approval prior to implementing each change? If so, describe the review/approval process.

SC9. Based on an accepted design, rapid prototyping is an approach used to quickly develop a working system upon which to build. Was this approach used on the subject system? If so, describe the process. If applicable, at what points were the evolving prototypes subject to review? Once approved, was each prototype frozen as a reference point from which successive changes were made?

SC10. Were there any controls placed on the software to track updated versions generated during system development? If so, describe the control mechanism.

SC11. A "good" AI system architecture may have the following characteristics:

- separate inference engine from knowledge base
- as uniform a knowledge representation as possible
- as simple an inference engine as possible
- overlapping knowledge

Describe the characteristics of the subject system. To what extent were the architecture characteristics of the subject system influenced by the following domain characteristics:

- size of solution space
- reliability of data or knowledge
- nature of data (dynamic vs static)
- nature of knowledge
- user interfaces

SC12. The following elements may be potential components of an ideal AI system (brief descriptions of each element are presented in Attachment 1):

- Language Processor
- Knowledge Base
  - Facts
  - Rules
- Justifier

- Interpreter
- Scheduler
- Consistency Enforcer
- Communications Handler

Which components does the subject system contain? What factors (e.g. design constraints, domain characteristics, cost/schedule constraints, etc) influenced the decision to include or exclude components? Was the architecture subject to review and approval by managerial and/or technical associates?

SC13. Please give an indication of the size of the AI system (number of rules in the knowledge base, lines of code or some other easily understood unit of measure). Identify the division between AI and conventional software components in addition to the percentage of code dedicated to each of the components listed below. Add components, if applicable to the subject system.

- knowledge base
- inference engine
- user interface
- support environment

SC14. Software tools are often imperative in AI system development because a small team is responsible for generating a large amount of code. Describe the software development environment under which the subject system was developed. If tools were not available, how do you think the lack of same hindered the project?

SC15. How was the AI software interfaced with conventional software? What problems were noted in making the interface?

SYSTEM EVALUATION/VALIDATION

SV1. Please describe your overall approach to evaluating the performance of the subject system. What type of test documents (e.g. plans, procedures, reports), if any, were written? Were users involved in the test process? Were test strategies applicable to conventional software such as multiple test levels (e.g. module test, integration test, program test), branch coverage, boundary value testing, stress testing, etc, used to test the AI system? If not, describe the strategy, if any, used to test the system.

SV2. If the knowledge base, inference engine, and/or other components exist as discrete entities in the subject system, were they tested separately? How did other architectural characteristics influence the test process?

SV3. Is the subject system self-modifying (is it capable of changing static evaluators, internal concept formulations, or modifying its inference engine in response to new data or incorrect reasoning)? If so, what impact did this characteristic have on your approach to testing? Did you repeat tests that performed properly prior to the self-modification? If not, how did you know that the modified software did not impact the performance of the system in a negative way?

SV4. It has been advocated that the evaluation of an AI system should emphasize quality assessment (ie overall user satisfaction) rather than more traditional performance measurements. In one particular investigation, an interactive evaluation subsystem was built into an expert system known as the Automated Academic Advisor[2]. The purpose of the subsystem was to monitor various parameters of system operation and to administer interactively questionnaires to the user while the application system was running. The subsystem would statistically analyze responses. Please state your opinion on this approach to evaluating system performance. Does the subject system contain such a built-in evaluator (or any other kind of evaluator)?

SV5. What kind of tools (built-in or otherwise) and techniques did you use to evaluate/validate the AI system?

SV6. When testing the subject system, what criterion did you use to judge whether the system passed or failed a test? Please elaborate for all levels of test (e.g. for testing a specific rule and for testing the system as a whole). What criterion was used to judge that the system was ready for use?

SV7. In rapid prototyping, the general approach is "build a little, test a little". If rapid prototyping was used for the subject system, please describe the method of system evaluation. Did the testing phase become more rigorous as the system matured?

FIELD SUPPORT

The following questions apply only to those systems that are actually being used in the field. If use of the subject system is limited to a research and development environment, proceed to the next section.

FS1. Was performance an issue, either in terms of response time or degree of accuracy? If applicable, describe the effort taken to improve system performance and its effect on phases of the project that had been frozen.

FS2. Were any enhancements suggested by the users as they learned to work with the system? If applicable, describe the mechanism for implementing the suggested changes.

[2] Cercone, N., et al, "Designing and Automating the Quality Assessment of a Knowledge-Based System: The Initial Automated Academic Advisor Experience", *IEEE 1984 Workshop On Principles Of Knowledge-Based Systems*, IEEE Catalog Number 84CH2104-8.

PART IV: MISCELLANEOUS

M1. Lessons learned - Looking back over the entire development cycle of the subject system, are there any activities that should have been done differently? If so, describe the activities, how they should have been done, and whether or not the new approach will be used on a future AI system.

M2. Please provide any additional information regarding characteristics of the software development you consider unique to the subject system that were not covered in your responses to any of the preceding questions.

ATTACHMENT 1

GENERAL DEFINITION OF TERMS

Language Processor. A language processor mediates information exchanges between the expert system and the user by processing questions, commands and volunteered information expressed in a problem-oriented language.

Knowledge Base. A knowledge base is a repository for rules, facts, and/or information about the problem to be solved.

- **Rules** - This part of the knowledge base contains procedural interpretations.
- **Facts** - This part of the data base contains declarative (i.e., non-procedural) information pertinent to the expert system domain.

Justifier. A justifier explains to the user why certain conclusions were reached and why others were not.

Interpreter. An interpreter executes the chosen agenda item by applying the corresponding knowledge base rule.

Scheduler. A scheduler, which may contain a fair amount of knowledge in its own right, controls the agenda by determining which pending action should be executed next.

Consistency Enforcer. A consistency enforcer attempts to maintain a uniform representation of the evolving solution by applying some quantitative scheme to determine the degree of belief in each decision.

Communications Handler. A communications handler manages the interfaces between components in a hybrid or multi-component system (e.g. blackboard).

Appendix C

Case Summaries

C.1 ARINC Summary

The ARINC System Testablility and Maintenance Program (STAMP), allows an engineer to analyze a system's testability for field maintenance operations. The system will recommend design changes, and provide fault isolation strategies for manual, semi-automatic or automatic field fault isolation.

The development team was comprised of four to five engineers, with no more than three computer programmers assigned at any one time. Two of the engineers had previous AI experience. Three of the project personnel had DOD software development experience. Nonetheless, only internal standards were used during the prototype and development phases.

STAMP was funded as an IR&D project. The initial prototype was released within 6 months followed by a field model which was developed within two years. The system's success is based on the fact that STAMP demonstrated the possiblity of a new technology and significantly increased productivity by automating a manual process. STAMP is currently an established product at ARINC.

There was no requirements analysis for the prototype. However, a requirements analysis was performed for the rehosted version of STAMP from an Apple to the HP-1000 operating system. The only formal software development procedures followed for the initial prototype were during problem formulation which encompassed some of the ideas behind identification (determining problem characteristics), conceptualization (finding concepts to represent knowledge) and formalization (designing structures to organize knowledge, i.e., tool building). Throughout the prototype phase, the capability of the system was continually assessed. During development of the rehosted version of STAMP, more rigid standards were followed. More attention was placed on finding ways to represent knowledge and formalisms were developed. Extensive documentation was required.

The knowledge acquisition is inherent to STAMP. A new knowledge base is implemented by a testing expert with each new use of the tool.

STAMP is comprised of the following components: a knowledge base, an interpreter, a consistency enforcer and a compiler. Tools used during development encompassed a Fortran 77 compiler/debugger and the HP-1000 operating system.

The STAMP system is written as conventional software in Fortran 77. The system feasibility was tested by individuals who checked code at the module and integration level. Users were allowed a two month trial period to shake out report problems with the system. A user conference was then held to determine final action items.

Overall, the system underwent five major architectural changes which allowed for incremental testing and iterative system development. Some of the lessons learned were to develop a simple, usable prototype excluding bells and whistles; document early and frequently; free the experts from

administrative burdens to allow them the time and opportunity to develop ideas; and encourage creativity during the prototyping stage.

C.2 Boeing Computer Services Summary

Boeing Computer Services reported on a Strategic Force Management Decision Aid which is a knowledge-based replanner. The inputs to the system are: a description of a previously created plan for the employment of strategic forces, and a description of an event which would require alteration to the plan. The system then determines a suitable modification to the plan which is presented to the user for review and approval.

The project was developed with internal funds by one engineer with previous AI experience. The first prototype was developed in six man-months, and the system is not currently fielded. Boeing Computer Services had previously built over 30 prototype AI systems. In terms of demonstrating the feasibility of a new technology, the system is a success. Overall user satisfaction and significant productivity gains were also substantial.

The model used to develop the Strategic Force Management Decision Aid consisted of the following steps:

- ELICITATION - Knowledge is acquired from information sources to produce an information base.
- ANALYSIS - The information base is analyzed and structured to produce a knowledge base.
- TESTING - The knowledge base content is tested to produce a case base.
- REFINING - The case base is refined to produce an expert knowledge base.
- COMBINING - Multiple expert knowledge bases in the same or related problem domains are combined to form a knowledge network.
- TESTING - The contents of the knowledge network is tested to produce a knowledge network case base.
- TAILORING - The knowledge network case base is tailored to the requirements of the specific customer to produce a delivered knowledge based system.
- RECORDING - The delivered knowledge based system may incorporate recording of information that is used as an information source for further elicitation.

These steps were iterated several times, with feedback from subsequent steps used to revise the results of previous steps. Three review sessions with the 'customer' were held which included demonstrations of the current level of system functionality and an exchange of information, ideas and concepts.

A requirements analysis phase was performed which consisted of discussion sessions with the 'customer' to identify types of problems requiring a solution which uses the replan approach. For each

problem identified, a suitable problem solving methodology was determined. The set of problems and their corresponding solution methodologies were used as a definition of requirements. User input to this process was considered very useful.

Four experts were involved in the process of knowledge acquisition which consisted of informal interviews. A negotiation process was used to resolve any differences. The experts were chosen based on their years of experience and the community acknowledgement of the person's status as an expert. The Knowledge Engineer was also an expert in the field and provided additional verification of the experts accuracy. A frame based, logic based and production rule representation was used to encode the knowledge. Ill-defined and incomplete knowledge is handled through the implementation of alternate paths through the production rules in a manner similar to default logic.

The tool KEE, operating on a Texas Instruments Explorer Lisp Machine was used to develop the system. A rapid prototyping approach was also used in the development process. The evolving prototype was subject to review on a continuous basis informally, and formally at about two-month intervals. There was no approval necessary for design changes, and no controls were used to track the updated versions of the system.

The components of the system include:

- Language Processor
- Knowledge Base
- Justifier
- Interpreter
- Consistency Enforcer

In terms of size, the knowledge base consisted of 50 productions rules, approximately 45 frames used as templates, and 8000 lines of LISP code. During execution of a typical scenario, the knowledge base grows to 300 or more frames and several hundred logic based facts.

A set of test scenarios was used to validate the system. No formal test plans were developed. The success criterion used was to examine the plan generated by the system and determine if it would be considered an acceptable plan by an expert.

C.3 Boeing Military Airplane Company Summary

The system reported on by the Boeing Military Airplane Company in Wichita, Kansas, is an Internal Research and Development (IR&D) project called *AI for Automatic Target Recognition (ATR)*. The purpose of the system is to reduce problems in the modern battlefield environment by improving present target recognizers through AI techniques. This long term research project is currently in the feasibility demonstration phase.

The Boeing ATR software development team included one lead engineer with approximately twelve years of conventional software research, development and additional personnel knowledgeable in

various areas including image processing. The lead engineer had also completed an in-house AI Associate Training course. All involved personnel had considerable experience with DOD and commercial software development. Standards for conventional software development had a significant influence on the AI software.

The general process used in the development of ATR to date consisted of the following phases:

- Identification - a study was undertaken to determine military product areas that were suitable for improvement via AI technology (i.e., ATR/image understanding)
- Conceptualization - conceptualization of problem characteristics derived throughout the identification phase into system capabilities (i.e., dealing with sensor fusion capability, incomplete or uncertain data)
- Formalization - considered a particularly valuable phase in that the expert system development tool was selected which determined the form of knowledge representation for the ATR system
- Implementation - gathering knowledge via literature surveys with minimal expert consultation
- Testing - validating the knowledge base and system behavior (i.e., demonstrating system concepts)

Each of the above phases is considered appropriate for the development of an AI system. However, it was perceived that as additional information regarding the problem was encountered, an iterative phase which produced a new prototype model for each reiteration was necessary to the development of the ATR software. The cyclic phase deviates from DOD and company standards in that requirements are not fixed.

Knowledge acquisition, which is never considered complete, consisted of a literature survey in the areas of image understanding. Documentation was prepared with known techniques, and algorithms for image understanding were reviewed by internal company personnel.

The knowledge representation method of frames and rules was determined by the type of data and available tools. The knowledge base continues to expand with informal reviews and a formal approval process. Tools used in developing the software were Knowledge Craft (in particular OPS-5) and LISP.

The major components of the system: a knowledge base, justifier and an inference engine, were largely determined by cost and schedule constraints. In addition, the architectural components were influenced by the tools available and were subject to review and approval.

The ATR system has not reached a stage in which extensive project performance can be evaluated. As the system reaches completion, and is ready for deployment, user satisfaction will be examined as well as productivity. The feasibility of the system is determined by the ability to demonstrate concepts as in the original proposal and the evolving concept definition. Every rule was triggered and fired. However, not every combination was tested. The system passed testing when a few image inputs produced the appropriate output result. In addition, the user interface is continually tested. As each prototype was released, the main testing target concentrated on those areas where changes had been made.

Overall, the ATR system, while not an *expert system* in the classic sense, has a domain which is considered much more complex than the typical expert system. As a research project, the reliance on literature that influences and causes continual changes to the requirements, results in an iterative process through all the development phases and ends in a new prototype. The flexibility of a research area such as the ATR would suffer without the ability to rework requirements to accomodate important data or changes to information.

C.4 Brattle Research Corporation Summary

Using both venture and contract funding, Brattle Research Corporation is building a generalized text extraction system focusing on business information. The sources of business information are online database services such as Dow Jones News/Retrieval and wire services such as Businesswire and PR-Newswire. The system has two basic functions: topic recognition/retrieval and extraction. The topic recognition side of the system has gone through a feasibility study, several prototypes and is being appled in staff study contracts. In terms of breadth and depth of retrieval, the intelligent topic recognition aspect has been found to be much more accurate than currently available keyword search techniques.

Several extraction applications have been tested, and a general extraction utility is now in the engineering design phase. User interface issues are still in a very early stage.

The development environment for the system is based on the Symbolics 3600 using Lisp. For portability reasons, Brattle Research is moving from Zeta-Lisp to Common Lisp. The company has built its own database management system, information retrieval language, and their own text analysis tools. The tools were a tremendous aid in facilitating rapid prototyping and conceptual breadboarding.

Brattle Research employs a relatively organized development strategy: define overall project structure, identify milestones and allocate budget vs. phase. Documentation includes conceptual system specifications and design documents which contain timetables and budgets. All documentation receives both managerial and technical review with feedback in terms of consensus rather than strict approval/disapproval. Weak areas identified by the review process were focal points during the prototyping phase.

The project team currently consists of seven people all with more than seven years experience in the AI field. The overall projected team size is estimated at twelve.

For each topic, the knowledge acquisition process is based on working with someone who is familiar with the way a topic is described in the literature. The knowledge engineer works to deduce a set of patterns that describes the topic. The knowledge base is then comprised of linguistic patterns associated with particular topics. The inference mechanism includes a variety of techniques collectively described as a simple form of discourse analysis. Specific methods include pattern recognition (including some signal processing strategies - i.e. treat text as a stream of symbols), and linguistic mechanisms such as chart parsing, categorization of syntactic elements and treatment of context-free grammars.

In terms of user involvement, existing customers have been providing feedback on the prototypes.

Before releasing its product, Brattle Research plans to form a scientific advisory board to comment on the system.

Brattle uses Symbolics' configuration management system augmented with its own code to control software distribution. One of the Symbolics machines is a dedicated file server which collects all incoming information in a central database repository.

Test data, which consumes 70 megabytes of memory, is based on articles from The Wall Street Journal, Electronics magazine, PR-Newswire, etc. To date, the testing criteria has been absolute accuracy. Both regression analysis and blind tests, i.e. - test sets that haven't been processed before - have been used.

The system contains code to automatically compile the abstract descriptions of grammars and documents into machine code for quickly searching text.

C.5 Carnegie Group Inc. Summary

The Carnegie Group Inc. reported on the DISPATCHER System which monitors and controls a factory floor Automated Materials Handling System. The system, a contractual project, took thirty-six man-months to deliver. The prototype release took twenty-one man-months. The system's success is based upon the product's feasibility, user satisfaction and productivity gains.

The development team consisted of three engineers of which two had prior AI software development experience. None of the engineers had any exposure to DOD software development standards.

The general development process for the DISPATCHER system consisted of the following standard phases:

- Identification - determining problem characteristics;
- Conceptualization - finding concepts to represent knowledge;
- Formalization - designing structures to organize knowledge (tool building);
- Implementation - encoding knowledge; and
- Testing - validating a knowledge base and system behavior.

However, the development process did deviate from the above methodology in two ways. First, knowledge acquisition was an additional phase that began after identification and before conceptualization. Secondly, the DISPATCHER System was developed incrementally which involved stepping through each phase iteratively until the system was sufficiently refined. The product was continuously updated up to and after the final installation.

Although it appears that a requirements analysis was not performed, a specification document was produced. The development effort was bounded by the document and by consultation with the purchaser. After installation the development was bounded by negotiation between the vendor and purchaser.

Knowledge acquisition consisted of information obtained from the specification documentation and from informal contact with the intended users. No domain experts were available for consultation on the domain.

The DISPATCHER System is comprised of the following components: a separate rule and fact knowledge base, an inference engine and a communications handler. Tools used during development included the OPS5 and a code-generator for external database routine. The use of OPS5 mandated separate fact and rule knowledge bases while domain considerations required a communications handler.

The DISPATCHER System's functionality was tested by the user. A simulation tool was built as a means by which the major system with which DISPATCHER communicates could be tested. Testing was considered complete on the basis of casual and random tests of the system functionality.

Overall, response time and accuracy were main concerns throughout the development effort. Users provided continuous functionality improvement suggestions to the engineers who attempted to incorporate their ideas. A recommendation to commit the user to a more detailed specification of the system was made. Also, the system architecture could have been more carefully defined and reviewed at an earlier stage in the system development.

C.6 Digital Equipment Corporation Summary

Digital Equipment Corporation (DEC) reported on XCON, an expert system that configures DEC computer systems. Specifically, XCON accepts a list of items from a customer order, configures them into a system, notes additions/deletions made,and prints out a set of detailed diagrams depicting the spatial relationships amongst the components. With the automation of a formerly manual task, XCON has decreased the number of costly configuration errors and significantly increased customer order processing speed.

The development of XCON began in late 1978 at Carnegie-Mellon University (CMU) where it was known as the R1 system. In early 1980, it was installed at the first DEC plant and used on a daily basis. By January 1981, DEC no longer required assistance from CMU in terms of maintaining and developing XCON. Since that time, XCON has become a mature system and is currently in the "production mode" phase.

It is difficult to discuss XCON without mention of the adjunct expert system XSEL (expert selling tool). By submitting orders to XCON, XSEL helps the DEC sales personnel interactively configure computer systems to prepare accurate quotes and match specific products to customer needs. XSEL and XCON share the same knowledge base and together contain more than 10,000 rules. Because the knowledge base is so very dynamic, (i.e. new components are frequently introduced existing components are often modified), an upgraded version of XSEL/XCON is released quarterly. Over the past few years, DEC has adopted a formal release procedure which consists of the following four phases:

- Planning;
- Development;

- System Quality Review (SQR); and
- Release.

The procedure is iterative from Planning through SQR. In the Planning phase, the workload is prioritized, checks are made in the OPS5 code for dependency levels and interim target dates established as a function of the scheduled release date.

In the development phase, knowledge acquisition occupies a large portion of the effort. Multiple experts are involved for each new product. Technical design reviews are held at the team level and implementation of the changes within the nine VAX cluster environment occurs. Correctness is verified using several internally developed tools which perform checks on such things as rule syntax and database entries. The VMS configuration management system (CMS) is used to control all source files. Testing on the changes made (similar to unit testing) is also performed during this phase.

In the SQR phase, a project test plan is established and executed against a large set (> 1000) of hypothesized customer orders. The testing criteria is wholly qualitative, namely, if there are no major problems that are foreseen to adversely impact the business, the new version of the system is approved. Following a successful SQR, the Release phase begins. The target DEC facilities receive a new tape of XSEL/XCON, an installation checklist, an installation systems management guide, an on-line summary of new parts and system functionality and, if necessary, a summary of significant problems yet to be resolved.

The project team currently consists of 35 staff divided into the following groups:

- Administration (3 people);
- User Support (5 people);
- Technical Support - (6 people); and
- Knowledge Engineering (21 people).

The Technical Support team deals with the non-OPS5 code - there are 5 conventional languages that comprise the system as well as many databases. The Knowledge Engineering staff is typically sub-divided into 2-3 teams whose responsibility is knowledge acquisition and representation, OPS5 coding, and testing. The project is run using a management team concept wherein the project manager makes very few decisions without consulting pertinent team members.

C.7 Expert Technologies, Inc. Summary

Expert Technologies, Inc. (ETI) reported on their PEGASYS system which is an expert system for the automatic pagination of yellow page directories. The system has three modes of operation: batch, review, and development. In the batch mode, PEGASYS provides automatic pagination using heuristics. In the review mode, the user can review the system's pagination for quality

control and interactively make corrections. The corrections can be made manually - graphics object composition, or they can be inputted into the knowledge base - changing/refining a rule to yield better pages. In the development mode, the user has the ability to design new rule bases and examine the resulting new products.

PEGASYS was an internally funded project and ETI's first AI system. The development team consisted of 7 to 8 engineers, all with LISP experience. Three members of the team also had prior AI experience.

Seven of the team members developed the first two prototypes. The first one was developed in one month and the second in two months. Eight team members developed the third prototype and the deliverable system. Each took three months to develop. The PEGASYS system was delivered in 69 man-months, three months ahead of schedule.

PEGASYS is considered a success in terms of demonstrating the feasibility of a new technology and overall user satisfaction. The system's most significant success lies in its flexibility which results in a system that can be extended and maintained easily.

A requirements analysis phase was performed which included user input. The tools KEE and Knowledge Craft, along with Xerox machines and TI Explorers were utilized in this phase. A functional specification was generated which described 90% of the eventual system's functionality. The requirements were frozen at the end of this phase.

Knowledge acquisition was achieved through verbal communication between the domain experts and the knowledge engineers. Extensive documentation and prototyping was performed during this process. The documentation was reviewed by the senior design engineer for approval, and the prototypes were integrated into the system upon approval. The choice of experts was obvious as there are only a few people who have special pagination expertise. Three experts were used and knowledge from either one was encoded and results approved by the other two. This process allowed for complete conflict resolution.

The knowledge representation method consists of a semantic network of frames. This method was chosen because of the close match it provided with the domain knowledge. Incomplete knowledge was encoded procedurally. The knowledge base expanded during the development process with approval by 2 senior engineers. PEGASYS allows for extension of its knowledge base on an as-required basis.

There were three major break points during the development process where new designs were encoded. This resulted in a complete reimplementation of the system at the end of the second break point. Approval for these changes came from the project manager and two senior knowledge engineers. An in-house proprietary configuration management approach was used to track updated versions of the system during development.

The PEGASYS system has the following characteristics:

- A simple inference engine
- Knowledge encoded in terms of semantic primitives
- Meta-rules encoded as "process" rules (procedures and frames/semantic objects)

- Consistency Enforcer

The architecture was reviewed and approved by top AI specialists in industry and academia. The user interface was done conventionally, and was one of the two largest parts of the system.

Development tools were used in building the prototypes. The tools contributed to development frustrations because of the expertise of the AI programmers. Consequently, development was moved entirely to LISP programming.

An acceptance test plan (ATP) for the system was written before the start of the project. This plan detailed acceptability with respect to:

- Productivity
- Efficiency
- Software maintenance and performance

System performance was evaluated relative to the current in-house systems (i.e. manual and semi-automatic pagination). In terms of field support, fine tunings of the system architecture were incorporated to enhance the performance of PEGASYS in terms of response time and degree of accuracy.

C.8 Frey Associates, Inc. Summary

Frey Associates, Inc. reported on the Themis TM Management Information System. Themis is a natural language processing system that answers English requests about information stored in a database.

The project team started with 6 staff members and eventually grew to 12 members. One person had previous AI experience. Since most of Frey's projects are system oriented, as opposed to Data Processing (DP) efforts, it is surmised that most of the software engineers had DOD conventional software development experience.

Themis was funded internally and followed a schedule defined by management. The first prototype was completed within 6 months. The product before beta fielding took approximately 6 man years with the final product encompassing about 10 man years.

The general process used in the development of Themis consisted of the following phases:

- Identification - determining problem characteristics
- Conceptualization - finding concepts to represent knowledge
- Formalization - designing structures to organize knowledge (tool building)
- Implementation - capturing knowledge
- Testing - validating a knowledge base and system behavior

Documentation was provided at the end of the Identification, Conceptualization and Formalization phases. The Implementation phase was divided into 2 stages, prototype and actual product. An iteration from Identification to Implementation occured at least once. From then on, iterations were primarily between Formalization, Implementation and Testing (4+). Testing was used to provide a basis for the user documentation of the system.

As a natural language system, Themis is different from an expert system in the approach used for knowledge acquisition. The goals of the system were limited to the inquiry characteristics of one data base from which both the requirements and user interface plans were derived. The user's main contribution to the system effort was in defining the user interface and the types of utilities needed to support the types of queries needed. The user was also valuable in defining types of queries for Themis.

The knowledge representation method used was rule-based since it was regarded by the design team as most appropriate to natural language processing. The main programming tool used was InterLISP which included the Programming Assistant, DWIM, Masterscope, a structure editor, and debugging tools. Source management tools were also used.

During the development process, a series of prototypes were generated. Major design changes were approved and implemented throughout.

Themis is comprised of the following components: Language Processor, Rule-Based Knowledge Base, Justifier, Interpreter, Scheduler, Consistency Enforcer, and a Communications Handler. The AI software was interfaced through mail boxes to the conventional software.

Themis is self modifiable in that when vocabulary and other new rules are introduced, Themis will evaluate contradictions and generate new rules to clarify these contradictions.

Test and regression procedures are in place for Themis which include correct as well as incorrect queries. The testing data base is continually enhanced by internal testers, customers and users. Due to the self modifying nature of Themis, tests are repeated without ending sessions to ensure consistent results. User involvement is considered important during this phase and throughout the lifetime of the system. Themis has built in testing capabilities along with automated error logging facilities.

Overall, the development process used by Frey Associates, Inc. has proven quite successful. If any phase were to be enhanced, it would be the testing phase to ensure completeness of the knowledge.

C.9 GTE Data Services Summary

GTE Data Services reported on their Central Office Printout Analysis and Suggest System (COMPASS) project. COMPASS is an expert system designed to diagnosis problems with GTE's no. 2 EAX switch. As input, COMPASS receives a data file which contains error messages produced by a particular switch. The output COMPASS generates consists of problem and fault identification and maintenance suggestions.

This was GTE Data Services first experience with an AI project. The project team consisted of two to seven people depending on the project phase. One team member had previous AI experience and another had limited AI experience.

C.9 GTE Data Services Summary

COMPASS was internally funded as a feasibility study. Management defined a schedule where the prototype would be built in 36 months and three months later the system would be in a limited field study.

The success criteria for COMPASS are: technology feasibility, user satisfaction, and productivity gains. COMPASS has been rated as a success in technology feasibility, as it has demonstrated the usefulness of expert systems for some telecommunications problems. Currently the system is in a limited field study and the users appear to be satisfied with the content and form of COMPASS output. Success in terms of productivity gains has not yet been determined.

The stages of development in the COMPASS project are as follows:

- Identification - this phase was emphasized due to the system's origin as a feasibility study. An internal technical note and a paper were produced in this phase.
- Conceptualization - In this phase several internal technical notes were produced.
- Formalization - A software control knowledge base was added to the system to track the various knowledge bases and method files.
- Implementation - Knowledge acquired from the expert was collected into several documents referred to as Knowledge Acquisition Rules (KAR) documents

The above phases were iteratively visited, especially early in the project. During these phases, internal reviews were held to report progress and to establish conventions among the various developers. A requirements analysis phase was not performed. Instead the system was bounded by the knowledge necessary to analyze the error data and create maintenance suggestions for a certain class of switch problems. In retrospect, the developers of the system feel that more attention should have been paid to requirements and to testing.

A single expert was utilized to provide domain knowledge. The guidelines used to select the domain expert were: must be recognized as an expert by his peers in other GTE telephone companies, must be articulate, must be enthusiastic about the project, and must be available for knowledge acquisition one week per month for at least three years. The knowledge engineers together with the expert created KAR documents. The knowledge acquisition process naturally ended when a particular switch problem was properly diagnosed and appropriate maintenance suggestions were provided for the test data under consideration. The development tool KEE was used to create the knowledge base which consisted of frames and production rules.

Rapid prototyping was not used very much in the development of COMPASS. Some of the developers feel that this was a definite shortcoming. To track updated versions of the system, a software control knowledge base was used. This controlled how the system was to be built from the various files and also how changes and additions were to be saved. Actual end users were not involved in the development process. End users were represented by the domain expert who was a supervisor of potential end users.

COMPASS is comprised of the following components: a language processor which was supplied by KEE, and knowledge bases - some containing facts only and some containing mostly rules. There is no justifier, interpreter, scheduler, or consistency enforcer in the system. Communications is not

part of the system but is handled by conventional software on conventional hardware for speed and efficiency.

Depending on the type of diagnostic task, COMPASS contains up to eighteen knowledge bases with over 1000 frames and 15000 slots. Five of the knowledge bases contain over 500 production rules and there are more than 500 LISP procedures in the system. The user interface is handled by a single knowledge base with 20 frames and approximately 50 LISP procedures. The support environment is handled by one knowledge base with 10 frames and 20 procedures.

To evaluate the COMPASS system, independent experts were asked to read the KAR documents and to assess the system's output given some input. Small test files containing error messages for one or more problems were created and used to validate the system. The test files emphasized rule-path coverage through the various COMPASS phases. Boundary value and stress testing was also done. This was accomplished using test files which contained a small number of messages and single problems. Regression testing was emphasized after modifications were made. Documentation of the testing phase was limited to describing general procedures and overall results. LISP procedures were written to perform the regressing testing in batch mode. Criteria for a test to be considered a success was: for a given input, did the appropriate rules fire and was the proper output produced? Appropriate and proper were determined from the KAR documents. Another performance issue was the ability of the COMPASS system to perform with the same accuracy as experts.

The product developers of the COMPASS system feel that some improvements in the design procedures should be made. There should have been more consultation with the potential end users of the system and a life cycle should have been defined. The product developers should have been involved earlier in the life cycle, rather than just the prototype developers. A high level design document should have been produced early in the life cycle and more attention should have been paid to traditional software engineering techniques (e.g. maintainability is more important than elegance and efficiency).

C.10 IBM Federal Systems Group Summary

IBM Federal Systems Group reported on their Fault Diagnosis and Resolution System (FDRS), which is a hardware failure diagnosis for the ground-based equipment of the Air Force's satellite command and control facility. The system diagnoses the failure down to the level for which there is a redundant component and recommends the proper replacement procedures.

FDRS was developed with independent research and development funds by 4 engineers, one of whom had previous AI experience. An initial problem definition phase was completed in approximately 3 man-months, and the first prototype was developed in an additional 3 man-months. The entire project took approximately 20 man-months to develop. The final prototype was tested in with the actual command and control software using a hardware simulation, but currently has not been embedded in the operational system.

FDRS satisfies the criterion of demonstrating the feasibility of a new technology. Productivity gains and user satisfaction have not been determined yet since the system is not installed in the operational environment. The system has satisfactorily met two additional criteria; they are: speed of execution, and integration into the main system.

C.10 IBM Federal Systems Group Summary

The development approach used to build the FDRS system consisted of the following steps:

- Identification
- Conceptualization
- Formalization
- Implementation
- Testing

The above steps were repeated to provide more depth of knowledge with each prototype. Documentation which was produced included a detailed description of the knowledge, design and code documentation. Informal design reviews and code inspections were held.

No requirements analysis was performed. Instead, the problem was initially bound by a general statement of wanting to prove the feasability of a knowledge-based approach for this problem. The second prototype was analyzed to determine what additional requirements would be included in the final version.

Knowledge was not acquired through a domain expert, rather the engineers studied the requirements document for fault diagnosis and recovery. Tables were constructed for each device indicating each possible fault, the symptoms of a failure, tests which should be performed, and recovery actions. The knowledge was represented as production rules.

The tools used in the development process varied with each prototype. For the first prototype a commercially available shell was used. The knowledge based system in the second prototype was developed using a small internal knowledge based system shell. This prototype interfaced with a Pascal simulation which was 1/3 of the code. The final prototype was JOVIAL code which was integrated into the command and control system. Each new prototype was considered a separate research effort and underwent the standard research approval process. Each developer was responsible for certain areas of the software. This individual was the only one authorized to make updates to his assigned area. All software followed a naming convention to track the current versions of the system.

The FDRS system consists of a knowledge base and an interpreter. A uniform knowledge representation was used, as well as a separate, simple inference engine. In terms of size, the knowledge base contains approximately 100 rules. The following is a list of the components and the percentage of the total system.

Component	Percentage
Knowledge Base	20%
Inference Engine	50%
User Interface	20%
Support Environment	10%

No formal test plans were developed, instead each path of the knowledge base was tested by going back to the original table of possible failures and testing to see if each failure path was followed in the expected manner. Unit tests were performed on each separate rule base, and integration testing was performed on those rule bases which interacted with each other. The FDRS system was tested within a simulation of the command control system.

C.11 Inference Corporation Summary (Authorizer's Assistant)

Inference Corporation reported on an Authorizer's Assistant system which was built for American Express, Inc. The system aids human authorizers in evaluating complicated charge cases to determine if the charge should be authorized.

The Authorizer's Assistant was developed by 8 people. The project personnel was broken down as follows: 1 Manager, 4 Knowledge Engineers, 2 System Engineers, and 1 Technical Writer, all with previous AI experience. The first prototype was developed in 15 man-months, and 72 man-months later the fielded system was completed.

In terms of demonstrating the feasibility of a new technology and overall user satisfaction, the system is considered a success. Productivity gains are in the process of being measured. Inference noted additional criteria for this system as the direct savings due to fraud loss reduction, and the less tangible improvement in customer relations which are promoted by the systems explanation facility.

The following are the steps utilized in Inference's development approach.

- Knowledge Acquisition
- Design
- Implementation
- Knowledge Engineering/Implementation
- System Interfaces
- Validation & Development Tools
- Commercial Systems Coding
- Expertise Validation & Refinement
- Acceptance

The Knowledge Acquisition and Design steps were performed iteratively to produce a prototype (Implementation). With the prototype complete, the steps of Knowledge Engineering/Implementation, Systems Interfaces, and Validation and Development Tools were performed and produced a complete system. From the complete system the steps of Commercial Systems Coding and Expertise Validation and Refinement were performed. When these steps were complete, the project was ready for the final step of Acceptance. Throughout this process, there were discussions and reviews with

the customer. Documentation that resulted included: a Design Document, Project Organization Charts, Schedules, Users Manual, and a Manager's Manual.

The requirements analysis phase performed was focused on the activity of knowledge engineering. Users were included as experts and a Design Document was produced. The process was bounded by continuous discussions and reviews with the customer.

The method used to acquire knowledge for the Authorizer's Assistant was as follows:

1. Watch authorizers work
2. Team review of cases off line with experts
3. On going process of Knowledge Base refinement
4. Review of code by experts
5. Review of system's behavior of real cases off line with experts
6. Doing the job and learning first hand

The knowledge acquired was represented using production rules. The system's knowledge base has expanded and the decision not to include additional knowledge was a managerial one determined by the constraints of the contract. Six experts were used, and they were chosen by the customer's internal review process which is based on service, productivity and accuracy of work. One expert was used predominately more, who settled inconsistencies and discrepancies.

Inference's ART development tool was used, along with Symbolics Common LISP. Users were involved throughout the development process. The prototype was a subset of the final system, it was not and never intended to be "throw away code". Design changes did occur over the course of development, but the fundamental design did not change. Approval for the changes was via the project manager and interaction with the customer, when needed. The system is connected to a IBM transaction system. This proved to be a very time consuming process. There was no formal software configuration management on the project.

The Authorizer's Assistant has a separate inference engine, a uniform knowledge representation, language processor, incremental compiler as an interpreter, a communications handler, and a symbolic scheduler. The knowledge base consists of 800 rules. The following is the division among the types of code:

- Knowledge Base 20%
- ART Inference Engine 40%
- User Interface 12%
- Environment 28%

No formal test procedures were developed. Code was written to partially automate the review process such that sets of test cases could be run in batch mode. Users were involved extensively

in the testing process. The expert system results were compared with the experts conclusions and found to be correct 97% of the time. Testing was done at the system level only. The system is not self modifying.

System performance was very important. Performance was measured along three axes: CPU, swapping and garbage. The code was "instrumented" to find out where CPU was being used and modifications were then made to various parts of the system. Inference feels that the performance improvements were made too late in the development process, it would have been easier and faster to have made design decisions all along that would have contributed to a high performance system. User suggestions were made, and negotiation was used to determine if they would be implemented.

The following is a list of Inference's lessons learned from this project.

1. More design should be done up front.
2. Performance monitoring should be done throughout the development process.
3. More test beds should have been developed for simulating various parts of the customer's systems to avoid the dependency on last minute testing.
4. Application of more "classic" software engineering techniques.
5. Use of management tools for actively tracking the project: (i.e. Gantt and pert charts and automated schedulers).

C.12 Inference Corporation Summary (Medical Charge Evaluation Control)

Inference Corporation reported on a Medical Charge Evaluation and Control (Medchec) system which evaluates medical claims and prioritizes them with regard to possible mischarging. Inference has had extensive experience building Knowledge Based systems; prior to developing this system they had built somewhere in the vicinity of 10 - 20 systems. Medchec is being developed under contract by 3 engineers, one with previous AI experience. The first prototype was built in 6 man months.

In terms of overall user satisfaction the Medchec system is so far considered a success. Inference feels this to be the most important criteria for success. As far as productivity gains and demonstration of a new technology the system is also considered successful. Inference noted that another criteria against which to measure success was the degree to which the developed approach can be easily extrapolated to other applications.

The following are the steps and products used in Inference's iterative development approach.

C.12 Inference Corporation Summary (Medical Charge Evaluation Control)

Steps	Products
Identification	Proposal
Conceptualization	Rule set description
Formalization	Data structures
Implementation	Code/prototype
Testing	Test case panel results

It is important to note that these steps were not performed once in a sequential manner. Rather, all steps except identification could be entered from the previous and/or succeeding steps. Reviews were held at approximately 1 month intervals. These reviews included high level overviews, demonstrations, reports produced from the expert system, and PERT charts of current progress.

A requirements analysis phase was performed in which the users, who were also the experts, participated. The experts were interviewed and asked how they would audit hundreds of claims if they had the time. This system does not mimic any current operations but instead performs a depth of analysis never done before. The users/experts completely defined the requirements of the system.

Knowledge was acquired from 3 experts during group discussions in which very little conflict arose. The knowledge was grouped as follows.

1. experience of past mischarging
2. expectations of mischarging patterns
3. suggestions by knowledge engineer

The information was documented in the form of taxonomy and English-like "rules". The knowledge was represented in frames and rules. The frames were used to index demons which computed patterns of repetition of a service and costs of a service. The frames were also used to linearize and combine the various assertions of interesting patterns for reporting purposes. The rules were used to detect each pattern of mischarging, one main rule per pattern. The knowledge base expanded as "recommended by the experts".

Development tools used were Inference's Automated Reasoning Tool (ART) and LISP on a Symbolics 3675. Rapid prototyping was also employed. The philosophy on the Medchec project was to build prototypes as a subset or framework of the completed system. From the start it was designed to be extensible and expandable both in performance and capability of supporting all types of knowledge and inferencing or reasoning techniques. The prototypes were not throw away systems. Many low level design changes occured with no approval necessary to implement changes. Also, there was no formal software configuration management on the project.

The Medchec system has a separate inference engine, a uniform knowledge representation of facts and rules, a communication handler, and a justifier which was used to flush obsoleted facts. The system size is small and uses forward and backward chaining reasoning. The initial knowledge base consists of 50 rules. The communication handler consists of 3000 to 4000 lines of LISP to handle

data parsing from the claims database on a S-mini computer. Also included in the system is about 140 schemata for loss types and diagnoses. The following is the division among the types of code.

- Knowledge base 30%
- Inference Engine 10% - enhancement of ART features
- User interface 20%
- Support environment 40%

No formal development plans were developed to test the system. The experts review the reports generated by the system to assess accuracy. The Medchec system incorporates feedback from auditors on confirmed mischarging to alter its ratings.

C.13 Lockheed Aircraft Service Company Summary (Expert Software Pricer)

Lockheed Aircraft Service Company (LAS) reported on an Expert Software Pricer (ESP) system for software costing. Included in the ESP system is a knowledge based Expert Sizer which assists in estimating the size of the software system being bid. Once determined, the size can be input into one of ESP's pricing models.

ESP was an internally funded project and the first AI project developed by LAS. A two person team developed the first prototype in 4 man-months. It took 12 man-months to complete the system. The schedule was defined by the funding that was allocated. Where applicable, LAS Software Engineering Procedures were followed.

LAS feels that the system is successful in terms of demonstrating the feasibility of a new technology and achieving significant productivity gains. The ESP system has demonstrated the capability of applying AI techniques to software costing while also providing savings in software bidding time and effort. In terms of user satisfaction, LAS is encouraging the use of the system within the Lockheed Corporation.

The development approach used to build the ESP system consisted of the following stages:

- Identification
- Conceptualization
- Formalization
- Implementation
- Integration
- Demonstrate/Test

C.13 Lockheed Aircraft Service Company Summary (Expert Software Pricer)

About 2 internal reviews per month were held during the development process. These reviews involved status reporting and demonstrations. A requirements analysis phase was performed which yielded a Software Requirements Specification document.

Knowledge was acquired from examining source code and documentation from completed software systems. The functions of the systems and their associated sizes were then documented. Knowledge acquisition is considered to be an ongoing process. The knowledge acquired was represented as frames with incomplete knowledge denoted by flagged dummy information.

The ESP system was developed using the Lockheed Expert System (LES) shell. LES includes a backward chaining, goal driven inference engine. The user interface included in LES was enhanced to make it more user friendly and accommodate the highly interactive nature of the Expert Sizer. In an attempt to limit the solution space, each type of software system to be sized (i.e. avionics systems) is represented as a separate knowledge base. Rapid prototyping was performed very early in the design conceptualization phase. Users helped identify bugs and made recommendations for improvement during the development process.

ESP is under configuration management using the VAX/VMS CMS system. A Software Change Request must be approved before any change can be made. Once the change is completed, a Software Change Description must be generated before the change is incorporated.

The LES shell provided the following elements:

- Language Processor
- Justifier
- Interpreter
- Scheduler
- Consistency Enforcer
- Communications Handler

The knowledge base consists of 77 rules with 700 lines of factual knowledge. The user interface and support environment written for ESP consists of 1200 lines of Pascal code.

There were no formal test procedures used to validate the ESP system. Systems with known size and costs were used as test cases. If the ESP estimates were within +/- 80% of the actuals, then the system results were considered acceptable.

Knowledge acquisition has been a difficult and time-consuming task. It is felt that a tool to allow users to input their site specific sizing data is needed both to alleviate the system developers from the knowledge acquisition task and to allow users to more easily customize the knowledge base. LAS plans to build such a tool this year.

C.14 Lockheed Aircraft Service Company Summary (Frequency Hopper Signal Identifier)

Lockheed Aircraft Service Company (LAS) reported on a Frequency Hopper Signal Identifier system which detects and characterizes frequency hopped signals. The AI component aids in signal identification. This was an internally funded project and the second AI project developed by LAS. One person, with prior AI experience, developed the first prototype in 9 months.

The system is considered a success in terms of demonstrating feasibility of a new technology. As far as significant productivity gains, the system has been successful for some but not all possible signals/signal environments.

A requirements analysis phase was performed which yielded a technical proposal and report describing the theory and implementation in detail. User involvement in this process was considered very beneficial.

Knowledge was acquired through a literature review and interviews with experts. The experts provided a small amount of very useful information. The knowledge acquisition is considered to be an ongoing activity. The knowledge acquired was represented in a temporal framework with confidence levels attached to most inferences.

The system was developed in Common LISP on a micro Vax. Iterative cycles of design and implementation were employed. The magnitude of changes was large with no prior approval necessary. Rapid prototyping was used to test feasibility of implementation and to identify shortcomings in design. The iterative cycle consisted of the following steps:

- Implementation
- Refine prototype
- Test to determine shortcomings
- Upgrade and expand prototype

The prototype was never frozen. User participation greatly influenced the development process.

The architecture of the system consists of a uniform knowledge representation and a simple inference engine. The system contains the following components:

- Knowledge base
- Justifier
- Scheduler
- Consistency enforcer

The size of the components listed below is in terms of the number of LISP functions:

- Knowledge base - 50

- Inference engine - 60
- User interface - 20
- Support environment - 40

No formal test procedures were developed. The testing strategies consisted of simulated real time response; i.e. how fast could a hopper signal be detected and characterized, also, how do a wide variety of noisy signal environments effect performance. The criteria for system tests was that of consistency and improved performance rather than pass/fail.

The system developer feels that taking the time to try to obtain a general knowledge of the application, and how conventional methods approach the problems can be counterproductive since it tends to channel thinking along conventional lines. This was also true to some extent in reviewing AI approaches.

C.15 Lockheed-Georgia Company Summary

The Lockheed-Georgia Company (LGC), under a research and development contract from the U.S. Air Force, is developing a Pilot's Associate system. The objective of the system is to provide pilots of single seat fighter aircraft a near real time on board support system. The system's jobs include monitoring the mission environment, evaluating each situation, and providing intelligence to the pilot on the current capabilities of his aircraft and the tactics deemed usable in a specific situation. The information provided to the pilot is analyzed against the mission or alternate mission objectives.

The project team consists of over 40 engineers. 65% of the team has had some previous experience in AI and/or Expert Systems. All of the engineers on the project have had prior experience developing software for the Department of Defense.

The Pilot's Associate is currently in the analysis stage. It is expected to take over 240 man-months to conduct the analysis, develop a simulation package, conduct two demonstrations, and deliver documentation to the customer under Phase I. Significant productivity gains are scheduled for Phase II, since knowledge base designs will have been established. Phase II completion is scheduled for February, 1989 and aims at developing real time processing of cooperative Expert Systems. The success of this project will be measured by two demonstrations of a portion of the overall task.

System development of the Pilot's Associate is based on the rapid prototyping of elements within the system components and integration builds around the mission manager executive which is a central system blackboard. Informal reviews are held prior to each integration build. In addition, formal design reviews are scheduled during the three year project.

The requirements analysis phase was completed thirteen months after the start of the contract. This phase included rapid prototyping to define requirements for each system component. System Specifications and Subsystem Description documents were produced for each component of the system

The knowledge acquisition in this project is accomplished via documented interviews with Air Force fighter pilots. These interviews are reviewed by a technical review board consisting of Lockheed,

Air Force, and contracted experts. Knowledge is also acquired from the following sources: flight engineers, A/C design specialists, manuals and Air Force studies. All facts shared between the developers are posted on an electronic bulletin board for review by other experts. All documentation is approved prior to incorporation into the technical database of the contract. Approval is required prior to each knowledge base change and is granted at the project management level. Users are included in the following aspects of the development process: reviews, knowledge acquisition, demonstrations of the prototype systems, and quarterly scheduling.

The development tools employed on the Pilot's Associate project are: ART, LES, and OPS5. All subsystem developers have a major commercial tool or a mature internally developed tool. An audit trail is being used to track design decisions/design rational. The design changes are approved by program management. The prototype releases are under contractor configuration control and are subject to control review prior to release for integration. Baselines are being used to "freeze in time" portions of the system. These baselines are under contractor control. The configuration audit trail is maintained by using a Code Management System on a DEC VAX System.

The Pilot's Associate project has several expert systems integrated into the product. The system architecture is characterized by the following items:

- Knowledge bases and inference engines are primarily separate
- Several knowledge bases must be "translated" to be understood by one another
- Most inference engines are simple
- Overlapping knowledge.

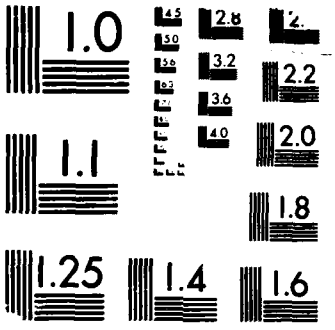
All subsystems contain a knowledge base, scheduler, and communications handler. A language processor and justifier are also components of the system. The mission manager handles consistency enforcement.

For system evaluation/validation the Pilot's Associate has developed and documented test plans. A Test Directorate, made up of experts and software engineers, develops test cases for formal review. The tests are conducted step by step and witnessed by personnel independent of the design personnel. A documented record is maintained for each test. Testing is rated as passed if the system requirements are satisfied. Experts will evaluate system performance against the specified performance of objectives and selected test cases. Currently no tools are being used in the testing phase. Quarterly demonstrations and reviews are conducted by management and customer personnel.

C.16 MITRE/Bedford Summary

MITRE reported on the Liquid Oxygen Expert System which was built for NASA Kennedy Space Center. The purpose of the system is fault detection and diagnosis of bad sensors and components in the liquid oxygen loading component of the Launch Processing System.

AD-A194 104 ARTIFICIAL INTELLIGENCE SOFTWARE ACQUISITION PROGRAM 3/3
VOLUME 1(U) SANDERS ASSOCIATES INC NASHUA NH
C BARDAWIL ET AL. DEC 87 RADC-TR-87-249-VOL-1
UNCLASSIFIED F30602-85-C-0254 F/G 12/5 NL



C.17 MITRE/McLean Summary

The project development team consisted of 2 people: one MITRE engineer with 6 years experience in building AI systems and one NASA engineer with no AI experience but considered the domain expert. Neither had experience with military software standards.

The Liquid Oxygen Expert System project was funded on a yearly basis and there was no predefined schedule. However, the first prototype was completed in 6 months and, after 2 years, the final prototype is being used in background mode. The system is considered a success in that it demonstrated the feasibility of a new technology and achieved overall user satisfaction.

The first 3 months of the project focused on the problem definition in which the users were indispensible equal partners. Requirements were never frozen, but rather evolved as did the team's understanding of the problem.

The knowledge acquisition process, which is never considered complete, involved informal discussions with NASA personnel, review of existing software and documentation that described the existing software. The knowledge representation method used was frame based since it best reflected the structure of the domain. Throughout the development of the system, the knowledge base did expand with the approval of the domain expert. Programming tools used were Symbolic's ZETALISP and FRL from MITRE and MIT.

During the development process of the final prototype, several "sub-prototypes" were built. Better ideas spurred major design changes which were then implemented as the basis of a new prototype. Neither designs nor prototypes were ever frozen; changes were made upon consensus of the domain expert. Because the domain expert is a user of the system, there was considerable user input throughout the development period.

The principal components of Liquid Oxygen Expert System are the knowledge base, a consistency checker, a fault location algorithm and an interface to the user or outside sensors. The components were chosen for appropriateness and decided by team consent. In terms of conventional software, the system interfaced to Modcomp minicomputers over a communications link for sensor data transmission.

The system was tested as it was developed, against test cases and live sensor data - its performance is judged by whether or not the users agree with its conclusions.

Overall, close user/domain expert/developer interaction had a strong and beneficial effect on system development. The absence of a formal development procedure was an extremely positive environmental factor, leading to a strong problem solving environment and high morale.

C.17 MITRE/McLean Summary

MITRE/McLean reported on the ANALYST system which was deployed to the 9th Infantry Division in Ft. Lewis, Washington. Further plans call for using the system as a test bed at both Ft. Sill and Ft. Leavenworth for DARPA projects through fiscal year 1988. ANALYST grew out of sponsored research to analyze and apply production system techniques to military applications. The effort evolved into a contract to develop a general knowledge-based aid for intelligence use. The system processes sensor returns from different intelligence sources and displays a transient situation of enemy combat units in real-time.

The project development team consisted of five people, including one domain expert, and one managing AI scientist. The estimated development time for the first prototype which did not emphasize performance was six to eight months. The following year and a half was devoted to enhancing system performance capability and building a knowledge base to facilitate the user modifications.

About eighteen months were spent doing requirements analysis. During this time, an A minus specification for ANALYST was produced. Requirements Analysis paralleled rapid prototyping and thus influenced the choice of AI technology and tools.

ANALYST was developed using in-house and LISP tools. The only tool purchased was a micro-compiler for the LISP machine. There were procedural differences between the development of ANALYST as an internal project and development for the field prototype. There were a number of observations made during development. It was recognized that the lack of standards was compensated for by the use of AI and LISP machines. Configuration control become more defined when the project started focusing on a delivery date. Control extended to files within a delivery build. Changes in code were documented with comments and verified by the programmer who made the modification. During development, the system was also used as a test bed for spatial and temporal techniques as well as evidential reasoning.

During the testing phase, a test of the inference engine and access to the knowledge and data bases was performed. A critical issue with the ANALYST system was the ability to interrupt during a test for a *what-if* question in order to explore the consequences of a particular path. The method tested for absences of firm requirements and complete specifications. Since several experts with different opinions were available, testing become even more difficult. However, the user community was conservative in their changes and considered the effects on the inference engine before suggesting a change. Regression tests were also performed to ensure that previous prototype capabilities were maintained intact.

Overall, the need for a prototype with user involvement in the demonstration or test phase was considered important. Once mature, it was stated that the prototype should be released for some limited operational testing after a much shortened in-plant acceptance test. Hardware should be subjected to the standard testing cycle. The need for a specification requirement for an explanation of the system's capability was recognized. The concept of training a domain expert about knowledge engineering was favored over teaching the knowledge engineer about the domain since the latter is a time-consuming factor. However, the knowledge engineer was considered to be the main implementor and director. In addition, the need for software discipline and standards was emphasized particularly with respect to the need for a specifications document as well as design and control principles.

C.18 Northrop/Aircraft Division Summary

Northrop/Aircraft Division reported on the Expert System for Target Attack Sequencing (ESTAS). ESTAS, an internally funded system, is a real-time system that provides decision aids for pilots under stressful and high-workload missions segment, i.e., the combat phase. Some of the decision aids are navigation verification, target prioritization, target selection, weapon allocation to targets,

and weapon readiness preparation. The system is expected to provide responses under all conditions including unpredictable ones.

ESTAS is designed to be one component of a highly integrated, flight simulation system. Its response is very dependent on the action/behavior of other components. At this time, however, schedule and funding constraints precluded the integration of ESTAS with the overall flight simulation system.

The ESTAS software development team consisted of four engineers, two with previous AI experience. In addition, two of the four engineers were experienced in conventional software development for DOD. The development of the system from problem definition to first prototype took six months or two man-years. Management predefined the schedule, and no follow-on effort to field the system followed.

ESTAS was developed within the framework of the traditional software development cycle with the exception that iteration amongst phases was acceptable. The specific model used to develop ESTAS consisted of the following phases:

- analysis - document the operational environment which included mission profiles and timelines;
- definition - document the system functional requirements and applicable areas for AI software development;
- identification - document specific input and output criterion, and general description of the application requirements based on problem characteristics;
- conceptualization - report on AI concepts and techniques that satisfy application requirements;
- knowledge acquisition - document rules and facts;
- formalization - expert system shell code and related documents;
- implementation - updated document of rules and facts as well as code the knowledge base; and
- testing - test result reports.

Iterations through formalization and testing were an inherent aspect of the software development effort. Requirements were frozen at the end of the analysis phase to avoid modification problems. Prototyping began after the conceptualization phase. Documentation was enforced but not based on the MIL-STD documentation practices. The documentation included software requirements, top-level design and detailed design. Technical reviews to assess system progress, as well as analyze and solve problems were held periodically. Progress reports were given to the developers. System demonstrations at the end of formalization, implementation and testing were also provided.

Knowledge acquisition consisted of domain research by the knowledge engineer and interrogations of the expert which included questionnaires, round-table discussions/interviews, and acting out of hypothetical situations. Laboratory simulators were used to illustrate the experts views and comments. The means used to gather the expert information involved note taking and tapes. Then, both were transformed into prose and eventually pseudo-code which underwent a formal

approval process. Knowledge acquisition was considered complete when the expert approved the knowledge base for the specific case. At this time, the knowledge base consists of 171 rules. The inference mechanism used is based on both forward and backward chaining.

The rule-based knowledge representation method was selected because of the need for real-time system responses. At this time there are no means for accomodating ill-defined or incomplete knowledge. The chosen representation was reviewed during a design walkthrough. The knowledge base continued to expand through informal reviews and a formal approval process until the first prototype release. The only tool used, the LISP workstation, incorporated required tools such as a language (Common Lisp), a graphics implementation and a developer interface.

The major components of the system are a knowledge base, a separate inference engine and a user interface. Cost and schedule constraints precluded the implementation of a language processor, a justifier and a consistency enforcer. The architecture underwent a formal review and approval by technical associates and the project engineer.

Performance of ESTAS was primarily based on user satisfaction and the expert's evaluation following incremental additions to the knowledge base or inference engine. Discrete components were not tested separately. Other than tracing facilities built into the system, tools were not used for testing. Because ESTAS was not integrated into the overall system, rigorous testing was not possible nor desirable.

Several observations were made throughout the development of ESTAS. First, system decomposition was believed critical to the successful integration of an AI system to a larger system. Second, it is believed that AI developments must go through the traditional system development cycle (iterations acceptable) involving the required engineering disciplines. The implication is that engineers across the various disciplines must be knowledgeable in AI. Third, the expert must be assigned and committed to the project. Expert participation on an "as available" basis hindered progress on ESTAS.

C.19 PAR Government Systems Corporation Summary

PAR Government Systems Corporation (PGSC) reported on three systems: Duplex Army Radio/Radar Targeting Decision Aid (DART), Cost Benefit of Tactical Air Operations (CBTAO), and See and Project Enemy Activity (SPEA). They are all decision aids for the Tactical Air Control Center, developed to the point of a working prototype. None of the systems have been deployed.

The DART Aid is designed to assist the Command, Control, and Communications Countermeasures (C^3CM) Analyst in the identification/classification of the following targets:

- Unidentified Command Post (UICP)
- Air Defense Regimental Headquarters
- SA-8 Battery
- SA-6 Battery

C.19 PAR Government Systems Corporation Summary

- SA-4 Battery/Battalion/Brigade
- Division Main Headquarters
- Division Forward Command Post
- Division Alternate Headquarters
- Radio Relay Stations
- ZSU 23-4
- EW Radars

The CBTAO decision aid allocates tactical air resources into mission packages, based on the highest probability of success. The system also provides the planners with estimates and explanations of the cost and benefit of such tactical mission packages. The planners using the system can accept or modify the recommended force packages based on his judgements.

The SPEA decision aid assists the tactical air resources allocation and employment process by providing planners with the ability to systematically project the dispositions of enemy forces up to 72 hours in advance. The force projections may be adjusted as necessary to compensate for perceived differences between the SPEA projections and the real world. The end product is a "planning picture" to be used by Combat Plans in interpreting the situation for the next day's Air Tactical Operations (ATO).

All three decision aids were contractually funded. DART was developed under the Decision Aids for Target Aggregation (DATA) RADC contract. Both CBTAO and SPEA were developed under the Integrated Tactical Air Control Center (ITACC) RADC contract.

At PGSC, one AI schooled engineer is tasked to work on a given decision aid. The AI engineer is team leader where a team can be composed of 3 to 5 computer scientists and one or more in-house domain experts. Essentially all the engineers have prior experience in developing software for the government.

The development process used to build the decision aids consisted of the following phases:

- Task I - Analysis and Design : preparation of Problem Definition Statement (PDS) and a high level design structure of the system. Includes requirements analysis wherein the domain experts define the problem;
- Task II - Development and Documentation : building a prototype or vertical slice of the system, and preparing technical, test and user documentation;
- Task III - Test and Evaluation : system evaluation, from a technical and operational standpoint.

The knowledge acquisition process was accomplished via sessions with knowledge engineers and government supplied experts. Knowledge gained was recorded either by tape or by hand. In-house experts or consultants were then used to verify the accuracy of the information provided. In cases

of questionable knowledge, a consensus approach was used. At PGSC, the preference is to train an in-house domain expert to be a knowledge engineer to work with the government experts rather than use a knowledge engineer with little or no domain experience. For the three decision aids, the knowledge was implemented in the form of production rules with confidence factors.

PGSC has internally developed several expert system shells that use a rule based, probabilistic inferencing mechanism. Each decision aid was developed using one of the in-house shells.

For each system, PGSC built a working prototype which, when coupled with the PDS, defined the final product and provided a proof in principle for the finished system. Users and experts were introduced to the system at the prototype stage, approximately 8 to 12 months into system development.

Both DART and CBTAO contain a knowledge base (facts/rules), justifier, rule compiler, inference engine, and scheduler. In the DART system the knowledge base contains 222 rules and 20,709 total lines of code. The DART system was developed using an in-house shell written in Pascal. The graphics and data base interfaces were written in C.

In the CBTAO system the knowledge base contains 50 rules and 12,886 total lines of code. The CBTAO aid was developed on the Symbolics 3600 and a Tektronix 4125 was used for color graphics display. A GKS package was developed to provide an interface to the Tektronix.

The SPEA system was built with object oriented programming using the Flavors package on the Symbolics 3600. The SPEA system is 6,000 total lines of code and contains 115 Flavors Definitions.

Formal Test/Evaluation Plans were developed for all three decision aids. The systems were tested as entities both technically and operationally. Technical evaluation was performed by a group of experts and nonexperts with computer science/ engineering backgrounds. Operational evaluation was performed by potential users who responded to questionnaires. The questionnaires were designed to extract the users' perception of the systems in terms of strengths, weaknesses and suggested improvements.

C.20 Sanders Associates, Inc. Summary

Sanders Associates Inc. reported on an internally funded effort to build an intelligent assistant for the task of reprogramming automatic test equipment. The purpose of TESS, the test assistant program, was to develop a technology base and capture some specific knowledge about testing of ECM systems.

There were three participants from the AI side of the house; experience was minimal, although two had successfully pursued academic studies in AI. The domain expert was from the user community. Substantially less than half time was available from the domain expert/user representative.

TESS personnel agreed with the steps outlined in the proposed development cycle. They stressed the iterative nature of the identification-conceptualization-formalization-implementation subcycle. This was driven predominantly by the need to generalize specific cases after enough problem understanding was achieved.

Products of the various stages included an IRAD plan in the identification phase, constraint definitions and user interfaces in the formalization phase, and knowledge, data layout language and attached primitives in the implementation phase.

Knowledge reorganization occurred within the FRL (Frame Representation Language) to accomodate generalization; sometimes this reorganization would require some functional extension in the FRL. The changes came from increased understanding of required functionality. The only controls on changes were the need for team agreement. The largest changes were in the area of the user interface and the knowledge base restructuring. It was felt that these areas represented something like exploratory programming.

Reviews were informal day long working sessions with AI project personnel, and a consultant. The main emphasis was on direction of future efforts with little time spent on reviewing past work.

The requirements analysis phase was used to set goals and directions. Many details of the requirements were deferred until prototyping was finished. No documentation was produced explicitly from the requirements analysis phase. There were IRAD plans, notes from review meetings, and the immediate code generation (i.e. documentation of system as built/executable). Requirements were not frozen but were left in abeyance. Users were included in the requirements analysis to the extent their project loading could spare them.

Testing was informal and done on small chunks of code. There was some reliance on repetitive execution to provide confidence in the reliability of the modules.

Tool usage was centered around the development system afforded by the Symbolics 3600 system. Some additional tools that came out of the TESS experience were FRL, the Data Layout Language (DLL), and the constraint propagation language (CPL).

Frames were selected as the basis of knowledge representation based on a consultants review of the project and its features. There was some support for incomplete or uncertain knowledge including decision postponement.

The TESS developers were not able to judge the completeness of the experts knowledge. They felt that there was not enough involvement by the expert. Several items that they would look for in an expert are: availability, commitment, interest, recent and on-going practice, competence, communication skills. Areas of interface and control with the expert were characterized by needing to establish a tasking or contractural relationship; periodic performance reviews, and shared physical accomodations.

C.21 SA&E Summary (Decision Support System)

SA&E reported on a decision support system which is a classified project. The prototype was completed within the first year. Subsequent releases are produced every three months. The system's development time has been reduced because of a tool built under the same contract, the Decision Support Development System (DSDS), which aids in the creation of run time systems.

The SA&E software development teams consist of two engineers to develop the decision support system and fifteen engineers assigned to DSDS. One of the engineers had extensive AI experience.

Also, one engineer had minimal experience with the Department of Defense software development process.

The general process followed in the development of the decision support system consisted of the following phases: requirements definition, specification, validation, generation, and verification. The requirements analysis performed involved a definition of goals, the impact of design options on schedule and the system architecture and an iterative redefinition of requirements for each incremental system release. Documentation consisted of a requirements specification that the customer and development manager reviewed and signed off for each increment, a design document for each iteration, and informal knowledge acquisition memos distributed to customers and the design team.

Knowledge acquisition involves regular interviews with the experts/users and additional research on decision support. Apparently, the incremental approach to software development helps. Experts are given a number of opportunities to supply knowledge information and to define the rules which govern the decision support system.

The knowledge representation method is rule-based. The DSDS environmental development tool used by the decision support system, makes use of Knowledge Engineering System (KES), Unify, LISP procedures and GE Scan.

The major components of the system are a knowledge base, a justifier, an inference engine, a language processor (LISP), a scheduler, an interpreter, and a communications handler (TCP/IP).

Alpha and beta tests are performed on the SA&E system. Alpha tests ensure that the integrated system works. It is then given to the application engineers for rigorous testing. Beta test involves a user demonstration of the system for evaluation and comments. Assessment of user satisfaction is accomplished via video records of customer reactions to the product. Acceptable system performance is determined by the experts.

Overall, the development and availability of DSDS vastly improved the decision support system's development time. Through the use of the DSDS environmental tool, a large production expert system can be more easily developed and maintained.

C.22 SA&E Summary (Sensitive Financial Analysis System)

SA&E reported on a sensitive system which was developed for an unnamed client. The system, which was contractually funded, analyzes financial data. The prototype was developed within three months followed by the final product eight months later.

The SA&E software development team included two engineers, of which one had previous AI software development experience. Neither engineer had previous experience with conventional software development for the Department of Defense.

The developers found the following software development model appropriate to the AI project:

- Identification - determining problem characteristics
- Categorization - categorizing the domain knowledge

- Structuring - form the framework for the knowledge by constructing the attribute hierarchy
- Implementation - capture knowledge
- Testing - validate the knowledge base and system behavior

A requirements analysis was performed. The system was built incrementally. Each increment consisted of a functional system with more capabilities. Informal reviews and approvals followed. No documentation was required by the customer. However, informal notes were distributed.

Knowledge acquisition involved the following process. First, specific statements were defined and used as a premise for judgements the system was expected to be capable of performing. Second, the data needed to produce the judgement was identified. Third, the logic required to make the connection between data and judgements was determined. Finally, case studies, with known outcomes were used to identifiy incomplete knowledge.

The knowledge representation method was rule-based. Tools used were the Knowledge Engineering System (KES), which is a high-level expert system shell, a standard text editor and the KES parser.

The major components of the system are a knowledge base, a justifier, a consistency enforcer, a communications handler and an inference engine. A selection of available software tools marketed by SA&E determined the system's architecture.

The system was evaluated by running a series of case studies with known outcomes through the system. Experts decided whether or not the system met the performance requirements. Both the system and the individual rules were tested.

Overall, the regular interactions between the experts and the engineers led to the success of the system. Experts were allowed an opportunity to familiarize themselves with the system, determine what kind of data the developing engineers needed, and to correct and expand the system.

C.23 Texas Instruments Inc. Summary

Texas Instruments Inc. reported on the Production Scheduler Expert System which was to provide a scheduling mechanism for textile fiber production and inventory. The system should have been commercially funded prior to each of the following three phases: the demo system phase; the prototype system phase; and the final production system phase. However, funding was removed three months before the prototype release due to a lack of customer resources. The estimated completion time for the final product phase was two years.

The Production Scheduler software development team consisted of one knowledge engineer to manage and implement the system and two domain experts that defined the knowledge and constraints. The knowledge engineer had a good working knowledge of AI tools and of Lisp programming techniques and conventional software development, but little AI software development experience. Other management and engineering resources were available. Minimal reviews between the knowledge engineer and the domain experts transpired. The meetings, although limited, were found extremely valuable.

No formal requirements analysis was performed. The requirement problems identified resulted from a previous attempt to resolve the problem through conventional programming techniques.

The general process used in the development of the Production Scheduler to date consisted of the following phases:

- Identification - determining the feasibility of the application based on the limited scope of the problem and the ability to gather a static collection of facts and constraints.
- Conceptualization - a frame based knowledge representation was chosen. Also, an initial knowledge network was defined.
- Formalization - designing structures to organize knowledge.
- Implementation - gathering knowledge via the user.

The structure of the knowledge base was modified several times throughout implementation. As previously mentioned, no prototype was released. Formal documentation was not provided at the end of the identification, conceptualization or formalization phases.

Minimal reviews between the knowledge engineer and the domain expert were allowed. The lack of direct access to the domain expert and the customer's management hindered the successful acquisition of knowledge. Also, incorrect assumptions were made by the knowledge engineer which resulted in disillusioned users. When the project was terminated, knowledge acquisition was not complete.

The knowledge representation method is frame based. The entire system development met with no formal approval process. Tools used to develop the software were LISP and a windowing forms management tool.

The major components of the system were an interpreter, a scheduler and a user interface. In addition, a communication handler had been planned.

A formal evaluation process, although scheduled, did not transpire since the system never reached completion. At the implementation phase, the domain expert checked the schedule test date for accuracy. The interface was tested separately. Then, both were tested together.

Overall, the customer's availability and committment as well as sufficient funds to complete the project were determined as necessary, but lacking for the Production Scheduler project. The relationship between the knowledge engineer and the domain expert cultivates the user's view of the system. Without regular reviews, the user's expectations of the system may not be satisfied. The ability to acquire knowledge from user's, and the customer's management should be encouraged. Also, acceptable system behavior should be documented.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END
DATE
FILMED
7-88
DTIC